

CUNOȘTINȚE DE SPECIALITATE

INFORMATICĂ ECONOMICĂ

CAP.1. BAZE DE DATE – Conf.univ.dr. ANCA MEHEDINȚU

**CAP.2. PROGRAMARE ORIENTATĂ OBIECT ȘI STRUCTURI DE DATE
– Prof. univ.dr. GEORGETA ȘOAVĂ și Conf.univ.dr. AMELIA BĂDICĂ**

CAP.3. SISTEME DE OPERARE – Conf.univ.dr. SORIN POPA

**CAP.4. PROIECTAREA SISTEMELOR INFORMATICE – Conf.univ.dr.
RĂZVAN BUȘE**

CAP.5. PROGRAMARE ÎN INTERNET- Conf.univ.dr. AMELIA BĂDICĂ

CUPRINS

CAP.1. BAZE DE DATE	3
1.1. CONCEPTE ALE BAZELOR DE DATE RELAȚIONALE	3
1.1.1. Definiții	3
1.1.2. Componente ale bazelor de date relaționale	3
1.1.3. Niveluri de abstractizare a datelor	7
1.1.4. Proiectarea bazelor de date relaționale. Etape. Normalizarea bazelor de date	8
1.2. CONCEPTE SQL	17
1.2.1. Conectarea la baza de date	17
1.2.2. Convenții de sintaxă SQL	19
1.3. DEFINIREA ȘI INTEROGAREA DATELOR FOLOSIND LIMBAJUL SQL	21
1.3.1. Instrucțiuni DDL (Data Definition Language)	21
1.3.2. Instrucțiuni DQL (Data Query Language)	26
1.3.3. Interogări din tabele multiple	33
1.3.4. Funcții SQL avansate	37
1.3.5. Folosirea avantajelor oferite de vizualizări	39
1.4. ACTUALIZAREA DATELOR FOLOSIND LIMBAJUL SQL	39
CAP.2. PROGRAMARE ORIENTATĂ OBIECT ȘI STRUCTURI DE DATE	43
2.1. CARACTERISTICI GENERALE ALE PROGRAMĂRII ORIENTATE OBIECT	43
2.1.1. Principiile ce stau la baza programării orientate obiect	43
2.1.2. Avantajele programării orientate obiect	45
2.1.3. Caracteristici generale ale claselor în Visual C++	47
2.2. UTILIZAREA BAZELOR DE DATE, PROGRAMAREA OLE ȘI COM	58
2.2.1. Utilizarea bazelor de date	58
2.2.2. Noțiuni de programare OLE și COM	59
2.3. TIPURI SI STRUCTURI DE DATE. TIPURI DE DATE ABSTRACTE	62
2.4. LISTE, STIVE, COZI	69
2.5. ARBORI. REPRESENTARE, PARCURGERE	73
CAP.3. SISTEME DE OPERARE	78
3.1. NOȚIUNI INTRODUCTIVE PRIVIND SISTEMELE DE OPERARE	78
3.1.1. Definiție și rol	78
3.1.2. Funcții	78
3.1.3. Componentele sistemelor de operare	79
3.1.4. Moduri de operare	81
3.2. PROCESE CONCURENTE	84
3.2.1. Noțiunea de proces	84
3.2.2. Stările unui proces	85
3.2.3. Planificarea unității centrale	86
3.2.3.1 Algoritmi de planificare a unității centrale	88
3.3. GESTIUNEA MEMORIEI	92
3.3.1 Gestiunea în cazul multiprogramării	92
3.3.2 Strategii de administrare a spațiului din memoria internă	97
3.3.3 Mecanismul de swapping	99
3.4. SISTEMUL DE OPERARE UNIX	100
3.4.1. Elemente introductive privind sistemul de operare UNIX	100
3.4.2. Gestiunea utilizatorilor și grupurilor	103
3.4.3. Permișiunile de acces asupra fișierelor și directoarelor	107
3.4.4. Sistemul de fișiere al UNIX	112
3.4.4.1 Partiții	112
3.4.4.2 Tipuri de fișiere	113

CAP.4. PROIECTAREA SISTEMELOR INFORMATICE	117
4.1. SISTEME INFORMATICE	117
4.2. METODOLOGII DE REALIZARE A SISTEMELOR INFORMATICE	119
4.2.1. Etapele de realizare a sistemelor informatice	119
4.2.2. Ordinea executării etapelor	120
4.2.3. Metode și tehnici de realizare a sistemelor informatice	122
4.3. ANALIZA SISTEMULUI INFORMAȚIONAL EXISTENT	124
4.3.1. Conceptul de analiză a sistemelor informaționale	124
4.3.2. Obiectivele și fazele analizei sistemului informațional existent (ASIE)	125
4.3.3. Organizarea și conducerea ASIE	126
4.3.4. Realizarea analizei sistemului informațional existent	127
4.4. PROIECTAREA GENERALĂ A SISTEMELOR INFORMATICE	134
4.4.1. Obiective și activități specifice proiectării generale	134
4.4.2. Proiectarea ieșirilor sistemului informatic	136
4.4.3. Proiectarea bazei informaționale de intrare	139
4.4.4. Proiectarea codurilor	142
4.5. PROIECTAREA DE DETALIU	145
4.5.1. Caracteristicile generale ale proiectării de detaliu	145
4.5.2. Proiectarea fișierelor	146
4.5.3. Stabilirea ordinii de prelucrare a fișierelor de bază	146
4.5.4. Determinarea procedurilor	147
4.5.5. Proiectarea procedurilor	147
4.6. IMPLEMENTAREA SISTEMELOR INFORMATICE	149
4.6.1. Asigurarea condițiilor de punere în funcțiune	150
4.6.2. Funcționarea experimentală și punerea în funcțiune a sistemului proiectat	151
4.6.3. Documentația finală a sistemelor informatice	152
4.7. EVALUAREA SISTEMELOR INFORMATICE	153
4.7.1. Eficiența economică a sistemelor informatice	153
4.7.2. Managementul calității sistemelor informatice	154
CAP.5. PROGRAMARE ÎN INTERNET	156
5.1. INTERNET ȘI WEB. ARHITECTURI DE SISTEME DISTRIBUITE	156
5.1.1. Noțiuni generale	156
5.1.2. World Wide Web	158
5.1.3. Protocolul HTTP	159
5.1.4. Arhitecturi multistrat	162
5.1.5. Paradigme de programare distribuită	163
5.2. PROGRAMARE PE PARTE DE CLIENT	166
5.2.1. Limbajul HTML	166
5.2.2. Tehnologii DynamicHTML	170
5.2.3. Miniaplicatii Java	173
5.2.4. Programare la client folosind scripting	175
5.3. PROGRAMARE PE PARTEA DE SERVER	178
5.3.1. Servere WWW	178
5.3.2. Gazde virtuale	179
5.3.3. Tehnologiile SSI și CGI	180
5.3.4. Miniservere Java	181
5.4. SERVERE DE BAZE DE DATE	184
5.5. LIMBAJUL DE METAMARCARE XML	188
BIBLIOGRAFIE	198

CAP.1. BAZE DE DATE

1.1. CONCEPTE ALE BAZELOR DE DATE RELAȚIONALE

1.1.1. Definiții

Baze de date

O bază de date este o colecție de informații interrelaționate gestionate ca o singură unitate. Această definiție este intenționat foarte largă, deoarece există mari diferențe între concepțiile diferiților producători care pun la dispoziție sisteme de baze de date. De exemplu, Oracle Corporation definește o bază de date ca fiind o colecție de fișiere fizice gestionate de o singură instanță (copie) a produsului software pentru baze de date, în timp ce Microsoft definește o bază de date SQL Server ca fiind o colecție de date și alte obiecte. Un *obiect* al bazei de date este o structură de date denumită stocată în baza de date, cum ar fi un tabel, o vizualizare sau un index.

Sisteme de gestiune a bazelor de date

Un *sistem de gestionare a bazei de date (DBMS - database management system)* este un produs software furnizat de producătorul bazei de date. Produse software precum Microsoft Access, Microsoft SQL Server, Oracle Database, Sybase, DB2, INGRES, MySQL și PostgreSQL fac parte din categoria DBMS sau, mai corect, *DBMS relaționale (RDBMS)*.

Sistemul DBMS pune la dispoziție toate serviciile de bază necesare pentru organizarea și întreținerea bazei de date, inclusiv:

- Transferarea datelor în și din fișierele fizice de date, în funcție de cerințe.
- Gestionarea accesului concurențial la date al mai multor utilizatori, inclusiv prevenirea conflictelor care ar putea fi cauzate de actualizările simultane.
- Gestionarea tranzacțiilor, astfel încât toate modificările făcute asupra bazei de date printr-o tranzacție să fie executate ca o singură unitate. Cu alte cuvinte, dacă tranzacția reușește, toate modificările efectuate de tranzacție sunt înregistrate în baza de date; dacă tranzacția eșuează, nici una dintre modificări nu este înregistrată în baza de date. Totuși, unele sisteme RDBMS nu asigură suportul pentru tranzacții.
- Acceptă un *limbaj de interogare*, care reprezintă sistemul de comenzi folosit de utilizator pentru a obține date din baza de date. SQL este principalul limbaj folosit pentru sistemele DBMS relaționale.
- Funcții pentru salvarea bazei de date și pentru refacerea bazei de date în urma erorilor.
- Mecanisme de securitate pentru împiedicarea accesului neautorizat la date și modificarea acestora.

Baze de date relaționale

O *bază de date relațională* este o bază de date care respectă modelul relațional, dezvoltat de Dr. E. F. Codd. Modelul relațional prezintă datele sub forma familiarelor tabele bidimensionale, similar cu o foaie de calcul tabelar Excel. Spre deosebire de o foaie de calcul tabelar, nu este obligatoriu ca datele să fie stocate într-o formă tabelară, iar modelul permite și combinarea tabelelor (*crearea uniunilor (joining)*, în terminologia relațională) pentru formarea vizualizărilor, care sunt prezentate tot ca tabele bidimensionale. Flexibilitatea extraordinară a bazelor de date relaționale este dată de posibilitatea de a folosi tabelele independent sau în combinații, fără nici o ierarhie sau secvență predefinită în care trebuie să se facă accesul la date.

1.1.2. Componente ale bazelor de date relaționale

1. Tabele

Unitatea primară de stocare a datelor într-o bază de date relațională este *tabelul*, care este o

structură bidimensională compusă din rânduri și coloane. Fiecare tabel reprezintă o *entitate*, ceea ce înseamnă o persoană, un loc, un lucru sau un eveniment care trebuie să fie reprezentat în baza de date, cum ar fi un client, un cont bancar sau o tranzacție bancară. Fiecare rând al tabelului reprezintă o apariție a entității.

Figura 1.1 reprezintă conținutul parțial al unui tabel numit **MOVIE** (Film).

Tabelul **MOVIE** este parte a unei baze de date pentru un magazin de produse video, folosită ca exemplu. Tabelul **MOVIE** conține date care descriu filmele disponibile în magazinul de produse video. Fiecare rând din tabel reprezintă un film, iar fiecare coloană reprezintă o caracteristică a filmului respectiv, cum ar fi titlul filmului.

<u>MOVIE_ID</u>	<u>MOVIE_GENRE_CODE</u>	<u>MPAA_RATING_CODE</u>	<u>MOVIE_TITLE</u>	<u>RETAIL_PRICE_VHS</u>	<u>RETAIL_PRICE_DVD</u>	<u>YEAR_PRODUCED</u>
1	Drama	R	Mystic River	58,97	19,96	2003
2	ActAd	R	The Last Samurai	15,95	19,96	2003

Fig. 1.1 Structura tabelii Movie (Film)

2. Relații

Relațiile reprezintă asocierile dintre tabelele bazelor de date relaționale. Deși fiecare tabel relațional poate exista independent, esența bazelor de date este tocmai stocarea informațiilor între care există legături. De exemplu, pe lângă filmele propriu-zise, se pot stoca și informații despre categoriile folosite de magazin pentru organizarea inventarelor de filme. În același timp, puteți stoca și informații despre copiile fiecărui film, inclusiv data la care a fost primită copia și formatul acesteia, cum ar DVD sau VHS. Prin folosirea relațiilor, se pot asocia tabelele înrudite, într-un mod formal, ușor de folosit astfel încât să combinăm date din tabele multiple în aceeași interogare a bazei de date, dar păstrând flexibilitatea de a include numai informațiile care îl interesează pe utilizator. Posibilitatea de a selecta din baza de date numai informațiile care ne interesează ne permite să ajustăm informațiile din baza de date în funcție de cerințele specifice ale persoanelor sau aplicațiilor care au acces la baza de date.

Figura 1.2 prezintă patru tabele din baza de date a magazinului de produse video și relațiile dintre acestea, într-un format cunoscut sub numele de diagramă de relații a entităților (ERD - Entity Relationship Diagram).

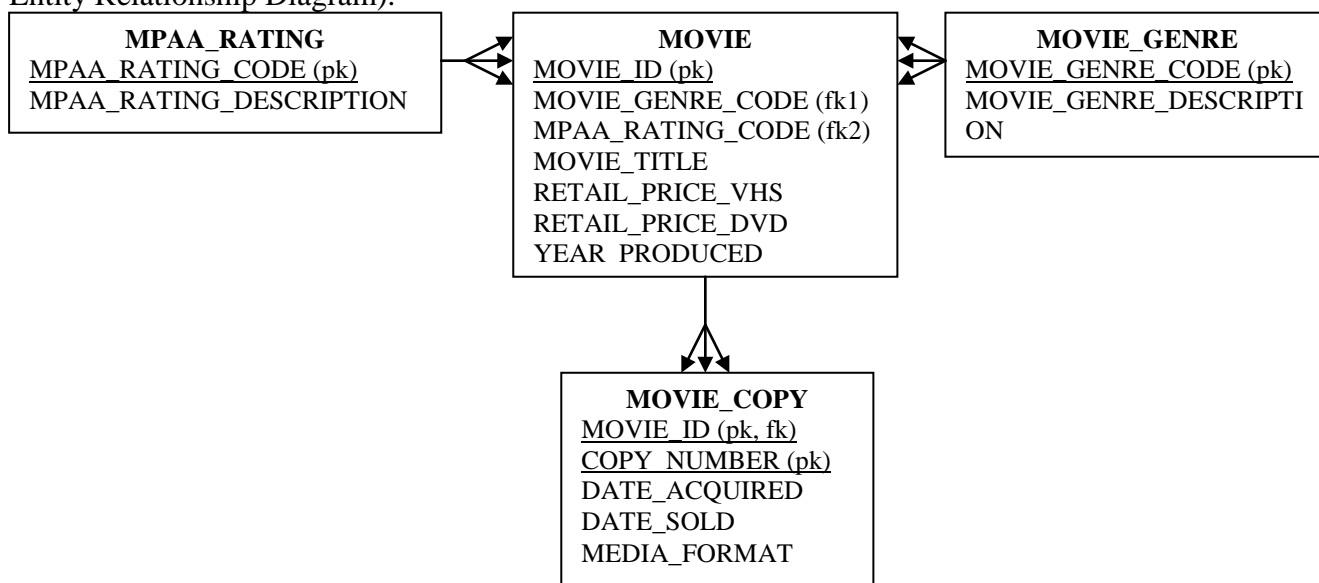
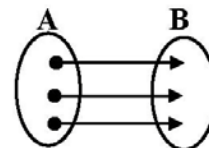


Fig. 1.2 Diagrama ERD a bazei de date pentru magazinul de produse video (prezentare parțială)

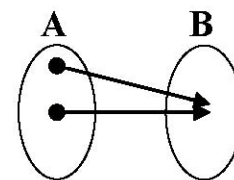
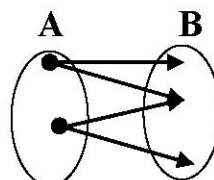
Diagramele ERD ne pun la dispoziție o modalitate de prezentare a proiectului general al unei baze de date relaționale, într-un format ușor de înțeles pentru utilizatorii bazei de date, indiferent dacă au sau nu cunoștințe tehnice. Fiecare dreptunghi din diagramă reprezintă un tabel relațional, cu numele tabelului scris deasupra liniei orizontale și coloanele tabelului enumerate pe verticală, în porțiunea principală a dreptunghiului.

În funcție de numărul de elemente, între care se stabilesc relații, aparținând celor două colecții, aceste relații pot fi de tip unu la unu, unu la mulți și mulți la mulți.

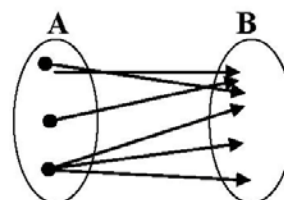
Relațiile de tipul 1→1 (unu la unu), care presupun că unui membru din colecția A îi corespunde un singur membru din colecția B.



Relațiile de tipul 1→m sau m→1 (unu la mulți sau mulți la unu), care presupun că unui membru din prima entitate A îi corespund mai mulți membri din a doua entitate B; astfel de relații se mai numesc și relații ierarhice.



Relațiile de tipul m→m (mulți la mulți), în care unui membru din entitatea A îi corespund mai multe date din colecția B și mai multor date din colecția A îi corespunde o singură dată din colecția B.



Relații de tip mulți la mulți se mai numesc și relații de tip rețea. O relație mulți la mulți se va descompune întotdeauna în două relații, o relație tip unu la mulți și respectiv o a doua relație de tip mulți la unu prin intermediul unei entități de legătură.

Fiecare relație este prezentată în diagrama ERD ca o linie ce conectează două tabele. Cele două capete ale liniei arată *cardinalitatea maximă* a relației, respectiv numărul maxim de rânduri dintr-un tabel care pot fi asociate cu un rând dat din tabelul aflat la celălalt capăt al relației.

Cardinalitatea maximă poate fi:

- *unu* (caz în care linia nu are nici un simbol special la capăt) sau
- *mai multe* (caz în care linia se termină cu un simbol numit *picior de cioară (crow's foot)* - linia se împarte la capăt în trei segmente).

În apropiere de capătul liniei se află un alt simbol, care arată *cardinalitatea minimă*, adică numărul minim de rânduri dintr-un tabel care poate fi asociat cu tabelul de la celălalt capăt al relației.

Cardinalitatea minimă poate fi *zero*, indicată printr-un cerc desenat pe linie, sau *unu*, indicată printr-o liniuță care taie linia relației.

De exemplu, relația dintre tabelele MPAA_RATING și MOVIE din figura 1-2 este o relație de tip *unu-la-mai-mulți*, ceea ce înseamnă că fiecare rând din tabelul MPAA_RATING (tabelul din partea „unu, numit și tabel *părinte*”) poate fi asociat cu mai multe rânduri din tabelul MOVIE (tabelul din partea „mai mulți” a relației, numit și tabel *copil*”), dar fiecare rând din tabelul MOVIE poate fi asociat cu un singur rând din tabelul MPAA_RATING.

Relația este logică, deoarece orice film lansat în SUA poate avea o singură categorie MPAA, dar o categorie poate fi asociată mai multor filme diferite. Este adevărat ca filmele sunt uneori „cenzurate” pentru a putea fi încadrate în diferite categorii, dar această problemă este rezolvată ușor, prin tratarea diferitelor versiuni ale aceluiași film ca și cum ar fi filme diferite, la fel cum facem atunci când un film este reluat cu alți actori. Este foarte important să ții seama de aceste lucruri, deoarece bazele de date relaționale acceptă numai relațiile de tip unu-la-mai-mulți sau unu-la-unu.

Cardinalitatea minimă arată dacă participarea la o anumită relație este obligatorie sau opțională. Toate relațiile din fig. 1.2 sunt obligatorii în partea „unu” și opționale în partea „mai mulți”, aceasta fiind cea mai frecvent folosită formă de relație. Dacă ne uităm din nou la relația

dintre tabelele MPAA_RATING și MOVIE, aceasta înseamnă că fiecare rând din tabelul MOVIE *trebuie* să aibă un rând corespondent în tabelul MPAA_RATING, dar nu este obligatoriu ca fiecare rând din tabelul MPAA_RATING să aibă asociat un rând din tabelul MOVIE. Dacă vrei să permiți ca inventarul de filme al magazinului să conțină titluri care nu au asociată o categorie MPAA, liniuța de la capătul dinspre tabelul MPAA_RATING al liniei care reprezintă relația cu tabelul MOVIE va fi înlocuită de un cerc. Deși sunt relativ frecvent întâlnite cazurile în care partea „unu” a unei relații nu este obligatorie, este foarte neobișnuit să aveți o relație în care să fie obligatorie partea „mai mulți” a relației, ceea ce ar însemna că tabelul părinte trebuie să aibă în orice moment cel puțin un „copil” în baza de date.

Relațiile sunt implementate folosind coloane corespondente din cele două tabele participante. În diagrama ERD, coloana sau coloanele subliniate din fiecare tabel, având în dreapta notația „pk”, reprezintă cheia primară (primary key), adică o coloană sau un set de coloane care identifică în mod unic fiecare rând dintr-un tabel.

Un tabel poate avea o singură cheie primară. Totuși, o cheie primară poate fi compusă din mai multe coloane, dacă aceasta este calea de formare a unei chei unice. Dacă o cheie primară este folosită într-un alt tabel pentru stabilirea unei relații, poartă numele de cheie externă.

În fig. 1.2, observați coloanele cheie externă folosite în tabelul MOVIE pentru crearea relațiilor cu tabelele MOVIE_GENRE și MPAA_RATING și marcate cu identificatoarele „<fk1>” și „<fk2>” în dreapta numelui coloanei cheie externă.

Cheile primare și cheile externe sunt blocuri de construcție fundamentale ale modelului relațional, deoarece stabilesc relații și permit crearea legăturilor între date, atunci când este necesar. Trebuie să înțelegeți acest concept pentru a putea înțelege cum funcționează bazele de date relaționale.

3. Restricții

O restricție este o regulă specificată pentru un obiect al bazei de date (de obicei, un tabel sau o coloană), având rolul de a limita într-un mod oarecare domeniul de valori permise pentru obiectul respectiv al bazei de date

După ce sunt specificate, restricțiile sunt impuse automat de sistemul DBMS și nu pot fi ocolite decât dacă o persoană autorizată le dezactivează sau le șterge (le elimină). Fiecare restricție primește un nume unic, astfel încât să poată fi referită în mesajele de eroare și în comenzile folosite ulterior în baza de date. Este recomandabil ca proiectanții bazei de date să denumească restricțiile, deoarece numele generate automat de baza de date nu sunt foarte descriptive.

Există mai multe tipuri de restricții pentru baze de date:

- **Restricția NOT NULL.** Poate fi plasată pe o coloană pentru a împiedica folosirea valorilor nule. O valoare nulă (null) este o modalitate specială prin care sistemul RDBMS tratează valoarea unei coloane pentru a indica faptul că valoarea coloanei respective nu este cunoscută. O valoare nulă nu este același lucru cu un spațiu liber, un șir vid sau valoarea zero — este o valoare specială care nu este egală cu nimic altceva.
- **Restricția cheie primară (primary key).** Definită pe coloana (coloanele) cheie primară ale unui tabel pentru a garanta că valorile cheie primară sunt întotdeauna unice în întreg tabelul. Atunci când cheia primară este definită pe mai multe coloane, combinația valorilor acelor coloane trebuie să fie unică în tabel - o coloană care reprezintă doar o parte a cheii primare poate conține valori duplicate în tabel. Restricțiile cheie primară sunt aproape întotdeauna implementate de RDBMS prin folosirea unui index. Indexul este un tip special de obiect al bazei de date care permite efectuarea căutărilor rapide în valorile coloanei. Atunci când în tabele sunt inserate rânduri noi, sistemul RDBMS verifică automat indexul pentru a se asigura că *pk* a noului rând nu este deja folosită în tabel și, dacă se întâmplă acest lucru, respinge cererea de inserare. Căutarea în indexuri se face mult mai repede decât căutarea în tabel; ca urmare, indexarea cheii primare este esențială pentru orice tabel, indiferent de dimensiunea acestuia, astfel încât căutarea cheilor duplicate la fiecare inserare să nu ducă la o reducere

semnificativă a performanțelor. O caracteristică suplimentară a cheilor primare este faptul că nu pot fi definite decât pe coloane pentru care a fost definită și restricția NOT NULL.

- **Restricția de unicitate (unique).** Definită pe o coloană sau un set de coloane care trebuie să conțină valori unice în cadrul tabelului. Ca și în cazul cheilor primare, sistemul RDBMS folosește aproape întotdeauna un index ca modalitate de impunere eficientă a restricției. Totuși, spre deosebire de cheile primare, un tabel poate avea definite mai multe restricții de unicitate, iar coloanele care participă la o restricție de unicitate pot conține (în cele mai multe sisteme RDBMS) și valori nule.
- **Restricția referențială** (numita uneori *restricție de integritate referențială*). O restricție care impune o relație între două tabele dintr-o bază de date relațională. Prin „impunere”, se înțelege că sistemul RDBMS se asigură întotdeauna, în mod automat, că fiecărei valori a cheii externe îi corespunde o valoare a cheii primare în tabelul părinte. Pe scurt, restricția referențială garantează că relația dintre cele două tabele și valorile corespondente ale cheii primare și cheii externe își păstrează logica în orice moment.
- **Restricția CHECK.** Folosește o instrucțiune logică simplă (scrisă în SQL) pentru a valida valoarea unei coloane. Rezultatul instrucțiunii trebuie să fie o valoare logică de adevărat (true) sau fals (false), astfel încât un rezultat adevărat să permită inserarea în tabel a valorii coloanei, iar un rezultat fals să ducă la rejectarea valorii coloanei, cu mesajul de eroare corespunzător.

4. Vizualizări

O vizualizare (view) este o interogare stocată în baza de date care pune la dispoziția utilizatorului un subset personalizat al datelor din unul sau mai multe tabele ale bazei de date. Cu alte cuvinte, o vizualizare este un tabel virtual, deoarece arată ca un tabel și, în cele mai multe privințe, se comportă ca un tabel, dar nu stochează date (nu este stocată decât interogarea SQL care definește vizualizarea). Vizualizările au mai multe funcții utile:

- Maschează coloanele pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează rândurile pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează operațiile complexe efectuate în baza de date, cum ar fi uniunile de tabele (respectiv combinarea coloanelor din tabele multiple într-o singură interogare a bazei de date).
- Îmbunătățesc performanțele interogărilor (în unele sisteme RDBMS precum Microsoft SQL Server).

1.1.3. Niveluri de abstractizare a datelor

Într-un sistem informatic ce utilizează BD, organizarea datelor poate fi analizată din mai multe puncte de vedere și pe diferite niveluri. De obicei, abordarea se face pe trei niveluri: *intern*, *conceptual* și *extern* (figura 1.3).

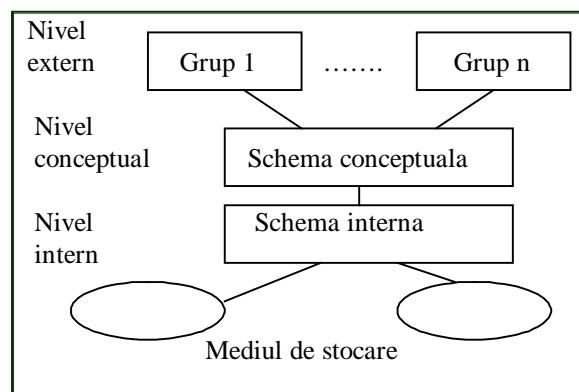


Fig. 1.3 Niveluri de abstractizare

Nivelul intern. Structura datelor este descrisă foarte detaliat, fiind accesibilă numai specialiștilor (ingineri de sistem, programatori în limbaje de asamblare sau alte limbaje apropiate

de „mașină”). Cele două părți principale ale bazei la acest nivel sunt:

1. un set de programe care interacționează cu sistemul de operare pentru îmbunătățirea managementului bazei de date.
2. fișierele stocate în memoria externă a calculatorului.

Fișierele ce conțin datele propriu-zise sunt alcătuite din articole sau înregistrări cu format comun. La acest nivel, structura BD se concretizează în schema internă.

Nivelul conceptual. Este nivelul imediat superior celui fizic, datele fiind privite prin prisma semnificării lor; interesează conținutul lor efectiv, ca și relațiile care le leagă de alte date. Reprezintă primul nivel de abstractizare a lumii reale observate. Obiectivul acestui nivel îl constituie modelarea realității considerate, asigurându-se independența bazei față de orice restricție tehnologică sau echipament anume. Întreaga bază este descrisă prin intermediul unui număr restrâns de structuri. Toți utilizatorii își exprimă nevoile de date la nivel conceptual, prezentându-le administratorului bazei de date, acesta fiind cel care are o viziune globală necesară satisfacerii tuturor cerințelor informaționale. La acest nivel, structura BD se concretizează în schema conceptuală.

Nivelul extern. Este ultimul nivel de abstractizare la care poate fi descrisă o bază de date. Structurile de la nivelul conceptual sunt relativ simple, însă volumul lor poate fi deconcertant. Iar dacă la nivel conceptual baza de date este abordată în ansamblul ei, în practică un utilizator sau un grup de utilizatori lucrează numai cu o porțiune specifică a bazei, în funcție de departamentul în care își desfășoară activitatea și de atribuțiile sale (lor). Simplificarea interacțiunii utilizatori-BD, precum și creșterea securității BD sunt deziderate ale unui nivel superior de abstractizare, care este nivelul extern. Astfel, structura BD se prezintă sub diferite machete, referite uneori și ca *sub-scheme*, scheme externe sau imagini (view-uri), în funcție de nevoile fiecărui utilizator sau grup de utilizatori.

1.1.4. Proiectarea bazelor de date relaționale. Etape. Normalizarea bazelor de date

Proiectarea unei baze de date este o activitate laborioasă și necesită parcurgerea următoarelor etape:

- formularea problemei;
- analiza cerințelor informaționale și definirea datelor de ieșire și a datelor de intrare;
- definirea tabelor, a structurii acestora și a relațiilor dintre table;
- optimizarea structurii bazei de date.

Odată ce acest proces a fost finalizat se continuă cu:

- proiectarea procedurilor tehnologice, pentru prelucrarea bazei de date;
- elaborarea programelor;
- testarea programelor;
- definitivarea documentației.

Toate aceste activități necesită, pentru proiectele reale complexe, o muncă în echipă pe baza unei metodologii riguroase, cunoscută ca metodologia de analiză și proiectare a sistemelor informatice. În cadrul unui sistem informatic baza de date reprezintă elementul central în jurul căruia se concentrează celelalte componente ale sistemului.

Formularea problemei presupune stabilirea obiectivelor aplicației informatice care asigură actualizarea și exploatarea bazei de date în concordanță cu cerințele managementului activității economice pentru care este proiectată baza de date. Obiectivele unei aplicații informatice sunt legate de asigurarea informațională a desfășurării proceselor decizionale specifice actului de conducere. Deci, noi trebuie să ne gândim că, prin existența unei baze de date, să asigurăm fondul de informații, într-o structură și de o calitate corespunzătoare cu cerințele managementului firmei. Baza de date trebuie să permită atât obținerea unor informații de detaliu, elementare, cât și calculul și prezentarea unor indicatori sintetici, agregați. Dacă am lua doar două obiective: reducerea costurilor și creșterea productivității muncii într-o firmă, atunci o bază de date trebuie să furnizeze informații despre consumul factorilor de producție,

costurile medii și globale, despre personalul muncitor și producția realizată, despre cheltuielile salariale etc. Aceste informații vor servi conducerii la identificarea căilor de reducere a costurilor și adoptarea celor mai adecvate măsuri pentru reducerea acestor costuri. După aplicarea măsurilor în practică, informațiile stocate în baza de date trebuie să permită de data aceasta și o analiză comparată a costurilor noi cu cele vechi, de exemplu, o analiză a dinamicii costurilor pe baza indicilor statistici. Am ales acest mic exemplu didactic pentru a accentua încă o dată complexitatea procesului de proiectare a bazei de date.

Analiza cerințelor informaționale, pornind de la obiectivele formulate anterior, se concentrează asupra a două probleme:

- indicatorii, rapoartele, listele și datele de ieșire care trebuie obținute;
- datele de intrare necesare pentru obținerea datelor de ieșire.

Acestea sunt cerințele informaționale care înglobează atât cerințele pentru datele de intrare pe baza cărora se creează și se actualizează baza de date, cât și cerințele pentru datele de ieșire folosite pentru urmărirea, controlul și dirijarea activității economice. Datele de intrare se culeg, de regulă, din documentele primare care circulă în cadrul fluxului informațional al firmei. Datele finale se vor integra în ansamblul de rapoarte, liste, situații cu rezultate pe care le furnizează sistemul informațional compartimentelor de conducere. Pentru indicatorii incluși în rapoartele finale, în general pentru oricare din indicatorii de ieșire, trebuie să fie foarte clar modul în care sunt obținuți prin prelucrarea datelor de intrare. În consecință, acolo unde este cazul, se precizează algoritmi de calcul, regulile de totalizare, sau alte reguli de obținere a fiecărei coloane, sau totaluri din rapoartele finale.

Aceasta are ca punct de plecare inventarierea câmpurilor prezente în situațiile finale și apoi gruparea lor în tabele. Gruparea câmpurilor pe tabele se realizează prin diverse metode. Dintre aceste metode, două sunt cele mai utilizate:

- analiza concordanței IEȘIRI – INTRĂRI;
- analiza semnificației semantice a datelor.

Analiza concordanței IEȘIRI – INTRĂRI este o tehnică specifică proiectării sistemelor informatice care identifică documentele primare din care se preiau datele de intrare folosite în calculul datelor de ieșire. Aceste documente vor constitui sursele de creare și actualizare a tabelelor bazei de date.

Tehnica este utilă în cazul în care proiectantul bazei de date este familiarizat cu sistemul informațional existent.

Un indicator prezent într-o situație finală se poate obține astfel:

- prin preluarea directă din documentul primar, respectiv din tabelul bazei de date;
- prin aplicarea unui algoritm de calcul.

Tabelele se compun prin gruparea câmpurilor pe principiul apartenenței acestora la anumite documente primare care circulă în cadrul sistemului informațional.

Analiza semantică se practică atunci când proiectantul bazei de date nu are la dispoziție un set de documente primare și apare în cazul sistemelor informaționale care se proiectează pentru firmele noi.

Definirea tabelor și relațiilor dintre tabele este etapa următoare în proiectarea bazei de date. Analiza cerințelor informaționale și a proceselor de prelucrare va conduce la identificarea datelor ce vor trebui stocate și care vor alcătui tabelele bazei de date. O tabelă va păstra datele fie despre toate caracteristicile unei colecții de date, fie numai pentru o parte dintre aceste caracteristici. Aici intervine spiritul analitic al proiectantului. Structura unei tabele este reprezentată de lista câmpurilor asociate tablei împreună cu descrierea atributelor fiecărui câmp (natură, lungime, număr de zecimale etc.). În structura unui tabel se regăsesc următoarele categorii de câmpuri:

- câmpuri de identificare (chei primare și chei condiționate);
- câmpuri tip dată calendaristică;
- câmpuri cantitativ-valorice;
- câmpuri de legătură cu alte tabele;

- câmpuri de stare care păstrează informații privind ultimele operații de prelucrare care au fost efectuate pe înregistrările din tabel.

Relațiile dintre tabele se caracterizează prin plasarea unor câmpuri comune în structura fiecăruia dintre tabelele aflate în relație directă. Pe baza acestor câmpuri, chei, fiecare sistem de gestiune a bazelor de date își construiește un mecanism propriu de accesare a înregistrărilor de date. Aceste mecanisme sunt transparente pentru utilizatorul obișnuit. Totuși este bine de reținut că nu orice câmp poate fi folosit la stabilirea unei relații, a unei legături între două tabele. Numai câmpurile de tip cheie candidat, care au proprietatea de a identifica în mod unic o înregistrare dintr-o tabelă, pot fi folosite în acest scop. Cheile candidate se mai numesc și indecși. Operația prin care se construiește sistemul de legături pentru ordonarea în vederea regăsirii înregistrărilor într-o tabelă se numește indexare. Prin indexare, fiecărei tabele principale de date i se va asocia o tabelă index corespunzătoare cheilor de indexare.

Optimizarea structurii bazei de date este un proces prin care se urmărește:

- reducerea redundanței datelor;
- eliminarea anomaliilor de actualizare.

Reducerea redundanței datelor până la un nivel minim și controlat urmărește eliminarea duplicării inutile a unor câmpuri în mai multe tabele sau eliminarea câmpurilor obținute prin calcul pe baza câmpurilor atomice. Un anumit nivel de redundanță, însă, trebuie admis pentru a nu denatura realitatea reflectată de date. De exemplu, câmpul **VALOAREA_CONTRACTULUI**, se calculează după relația: **VALOAREA_CONTRACTULUI=CANTITATE*PRET**

Nu este recomandată eliminarea acestui câmp pe considerentul că el se obține automat prin calcul. *De exemplu*, în cazul în care după un anumit interval de timp, prețurile suportă o majorare globală, cum se practică foarte des, atunci automat se vor modifica și valorile contractelor încheiate anterior datei de majorare a prețurilor ori acest lucru nu este corect, contractul odată perfectat nu-și poate modifica prețul convenit prin negociere.

Anomaliile de actualizare se referă la anomaliile de ștergere, respectiv de modificare. De exemplu, se consideră o firmă care derulează lunar mii de contracte de aprovizionare, pentru un nomenclator foarte mare de produse agroalimentare, dar care operează numai cu câțiva furnizori. Dacă în tabela **CONTRACTE** sunt incluse alături de codul furnizorului și denumirea furnizorului, contul său bancar și denumirea băncii, atunci va apărea următoarea anomalie de actualizare, în cazul schimbării băncii și a contului bancar de către furnizor. Acest lucru necesită parcurgerea tuturor înregistrărilor în care există aceste valori și actualizarea acestora (figura 1.4).

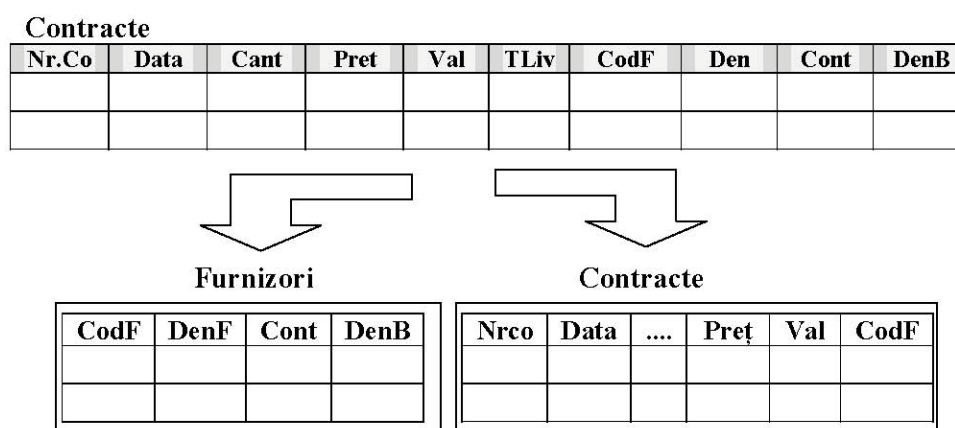


Fig. 1.4 Eliminarea anomaliilor de actualizare

Soluția este de a crea o tabelă separat, care are în structură: codul furnizorului, denumirea, contul și banca acestuia, în tabela **CONTRACTE** păstrându-se doar codul furnizorului ca element de legătură, ceea ce previne pierderea de informații prin spargerea unui tabel în două. În acest fel modificarea se realizează numai asupra unei singure înregistrări.

În 1972, Dr. E. F. Codd, părintele bazelor de date relaționale, și-a dat seama că tabelele

relaționale care îndeplinesc anumite criterii pun mai puține probleme la inserarea, actualizarea sau ștergerea datelor. Ca urmare, a pus la punct un set de reguli care trebuie respectate (organizate în trei „forme normale”) și un proces numit normalizare, care este o tehnică pentru producerea unui set de relații (termenul folosit de Dr. Codd pentru tabele) cu proprietățile dorite.

Necesitatea normalizării

Figura 1.5 prezintă tabelul **MOVIE** fără normalizare, așa cum ar arăta dacă toate informațiile despre filme ar fi colectate într-un singur tabel. Acest exemplu va fi folosit pentru ilustrarea procesului de normalizare.

În general, numele coloanelor din tabelele relaționale folosesc liniuțe de subliniere pentru separarea cuvintelor. În discuția despre normalizare am eliminat aceste liniuțe din figuri, pentru a face textul mai ușor de citit.

Există trei probleme care pot apărea în tabelele fără normalizare din bazele de date relaționale și toate trei există și în tabelul din figura 1.5. Scopul procesului de normalizare este de a elimina aceste probleme (anomalii) din proiectul bazei de date.

MOVIE_ID	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	LANG_CODE	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT
1	Drama	Drama	en,fr	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD
2	ActAd	Action and Adventure	en,fr,es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD
2	ActAd	Action and Adventure	en,fr,es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		VHS
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	1/10/2005	1/30/2005	DVD
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	2/15/2005		DVD

Fig 1.5 Tabelul MOVIE fără normalizare

Anomalia de inserare

Anomalia de inserare se referă la o situație în care nu puteți insera date în baza de date din cauza unei dependențe artificiale dintre coloanele unui tabel. **Să presupunem** că vreți să adăugați în baza de date a magazinului de produse video un nou gen de film (în coloanele **GENRE_CODE** și **GENRE_DESCRIPTION**) care urmează a fi folosit pentru clasificarea filmelor. Modelul din figura 1.5 nu permite acest lucru decât dacă aveți un film care să fie plasat în categoria respectivă și pe care va trebui să-l adăugați în tabelul **MOVIE** în același timp. Aceeași restricție este valabilă și pentru coloanele **MPAA_RATING_CODE** și **MPAA_RATING_DESCRIPTION**. Ar fi mult mai bine dacă ați putea adăuga noile genuri și categorii înainte de primirea filmelor în magazin.

Anomalia de ștergere

Anomalia de ștergere este inversul anomaliei de inserare. Se referă la situația în care ștergerea unor date duce la pierderea neintenționată a altor date. **De exemplu**, dacă primul film din figura 1.5 (*Mystic River*) este singurul rând din tabelul **MOVIE** pentru care coloana **GENRE_CODE** are valoarea „Drama” și este șters, se pierde informația că a existat vreodată un gen numit „Drama”.

Anomalia de actualizare

Anomalia de actualizare se referă la o situație în care actualizarea unei singure valori necesită actualizarea mai multor rânduri. De exemplu, dacă în tabelul prezentat în figura 1.5 trebuie să modificați descrierea codului **MPAA_RATING_CODE** „R”, trebuie să modificați și toate rândurile din tabel pentru filmele cu codul respectiv. Probleme similare apar și pentru coloana **GENRE_DESCRIPTION**. Chiar și coloana **RETAIL_PRICE** are această problemă,

deoarece toate copiile aceluiasi film (cu aceeași valoare MOVIE_ID) pe același mediu (DVD sau VHS) ar trebui să aibă același preț. Un alt pericol legat de această anomalie este faptul că stocarea unor date redundante poate duce la posibilitatea de a actualiza numai o parte a copiilor respectivelor date, ceea ce ar avea ca rezultat apariția inconsecvențelor în baza de date.

Aplicarea procesului de normalizare

De obicei, normalizarea începe de la mijloacele de redare a datelor care sunt (sau vor fi) prezentate utilizatorilor, cum ar fi pagini web, ecrane ale aplicațiilor, rapoarte și așa mai departe. Colectiv, acestea sunt numite *vizualizări de utilizator (user views)*. Poate părea ciudat la prima vedere, dar este ceva obișnuit ca proiectarea unui sistem de prelucrare a datelor să înceapă de la rezultatele pe care le va vedea utilizatorul, parcurgând apoi drumul înapoi către mijloacele folosite pentru obținerea rezultatelor dorite.

În timpul proiectării bazei de date, procesul de normalizare este aplicat fiecărei vizualizări, iar rezultatul este un set de relații normalizate care pot fi apoi direct implementate ca tabele ale bazei de date relaționale. Procesul în sine este destul de simplu, iar regulile nu sunt foarte dificile. Totuși, stăpânirea procesului de normalizare cere timp și exercițiu, în special deoarece impune proiectantului să se gândească într-un mod conceptual la datele și relațiile pe care intenționează să le folosească.

În timpul normalizării, considerați că fiecare vizualizare este o relație. Cu alte cuvinte, conceptualizați fiecare vizualizare ca și cum ar fi deja un tabel bidimensional - un mod de lucru pentru care aveți nevoie de experiență.

Rețineți că scopul procesului de normalizare este eliminarea anomaliilor de inserare, actualizare și ștergere. Procesul determină crearea unui număr mai mare de relații decât ați avea într-un model fără normalizare. Relațiile suplimentare sunt necesare pentru eliminarea anomaliilor, dar împărțirea datelor în mai multe relații face ca extragerea datelor stocate să fie puțin mai dificilă. De fapt, sacrificați o parte din performanțele de extragere a datelor și din ușurința utilizării pentru ca operațiile de inserare, actualizare și ștergere să fie mai simple.

Alegerea unui identificator unic

Primul pas al procesului de normalizare constă în alegerea unui *identificator unic (unique identifier)*, care este un atribut (o coloană) sau un set de attribute care identifică în mod unic fiecare rând de date dintr-o relație.

Identificatorul unic va deveni ulterior cheia primară a tabelului creat din relația normalizată. Pentru normalizare, *este obligatoriu ca* fiecare relație să aibă un identificator unic, în multe cazuri, puteți găsi un atribut care identifică în mod unic datele din fiecare rând al relației pe care vreți să o normalizați. Atunci când nu puteți găsi un singur atribut care să poată fi folosit ca identificator unic, este posibil să găsiți mai multe attribute care pot fi concatenate (combinat) pentru a forma un identificator unic. Atunci când identificatoarele unice sunt formate din attribute multiple, fiecare atribut rămâne pe propria lui coloană - nu faceți decât să definiți un identificator unic format din mai multe coloane.

În foarte puține cazuri, într-o relație nu există un set rezonabil de attribute care să poată fi folosit ca identificator unic. Atunci când se întâmplă acest lucru, trebuie să inventați un identificator unic, deseori cu valori atribuite secvențial sau aleatoriu pe măsură ce noile rânduri de date sunt adăugate în tabelul bazei de date. Această tehnică este sursa unor identificatoare unice, precum numărul de asigurări sociale folosit în Statele Unite, numerele de identificare ale angajaților, numerele de înmatriculare ale mașinilor sau CNP.

Relația Movie din figura 1.5 ne pune o problemă în privința găsirii unui identificator unic. La prima vedere, ar părea că atributul Movie_ID este cel mai potrivit în acest scop. Totuși, observați că valorile Movie_ID „2” și „3” apar de câte două ori, așa că, fără nici un dubiu, această valoare nu este unică. Problema este că valoarea Movie_ID identifică în mod unic fiecare titlu, dar magazinul urmărește separat fiecare copie a filmelor pe care le are în stoc. Cauza este faptul că magazinul se ocupă și de închirierea filmelor și vrea să se asigure că fiecare client returnează exact copia pe care a închiriat-o. După inspectarea datelor folosite ca exemplu și o scurtă discuție cu proprietarul magazinului, ajungeți la concluzia că în relația Movie nu există

nici o combinație de atribute care să identifice în mod unic fiecare copie a unui film, așa că inventați un atribut numit Copy_Number și-l adăugați în relație. Ori de câte ori inventați un identificator unic (sau o parte a unui identificator) este foarte important ca toți să înțeleagă valorile pe care le va lua acest identificator.

În acest caz, proprietarul magazinului decide ca valorile Copy_Number să reînceapă de la 1 pentru fiecare valoare Movie_ID, ceea ce înseamnă că valorile Copy_Number sunt unice numai în combinație cu valorile Movie_ID. Relația rezultată este prezentată în figura 1.6.

Prima formă normală: eliminarea datelor repetate

O relație este în *prima formă normală* atunci când nu conține atribute cu valori multiple (atribute multivaloare), adică atribute care au mai multe valori pentru același rând de date. Într-o relație, orice intersecție a unui rând cu o coloană trebuie să conțină *cel mult* o valoare pentru ca relația să fie în prima formă normală. În figura 1.6, atributul pentru limbă (Lang. Code) conține mai multe valori pentru unele dintre filme, așa că-l puteți considera un atribut cu valori multiple.

MOVIE_ID (pk)	COPY_NUMBER (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	LANG_CODE	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT
1	1	Drama	Drama	en,fr	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD
2	1	ActAd	Action and Adventure	en,fr,es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD

Fig. 1.6 Relația Movie după adăugarea atributului Copy_Number

Pentru transformarea relațiilor ne-normalizate în prima formă normală, trebuie mutate atributele multivaloare și grupurile repetitive în noi relații.

Procedura de mutare a unui atribut multivaloare sau a unui grup repetitiv într-o nouă relație constă în următoarele etape:

1. Creați o nouă relație, cu un nume sugestiv. Deseori, este bine să includeți numele relației originale, parțial sau în întregime, în numele noii relații.

2. Copiați identificatorul unic din prima relație în noua relație. Datele depind de acest identificator în relația originală, așa că trebuie să depindă de aceeași cheie și în noua relație. Identificatorul copiat va deveni cheie externă în noua relație.

3. Mutați grupul repetitiv sau atributul multivaloare în noua relație.

4. Formați un identificator unic în noua relație, adăugând atribute la identificatorul unic copiat din relația originală. Ca întotdeauna, asigurați-vă ca identificatorul unic nou format conține numai numărul minim de atribute necesar pentru a-l face unic. Dacă mutați un atribut multivaloare, care, în esență, este un grup repetitiv cu un singur atribut, este adăugat atributul respectiv pentru formarea identificatorului unic. Poate părea ciudat la prima vedere, dar identificatorul unic copiat din relația originală nu este doar o cheie externă, ci, de obicei, și o parte a identificatorului unic (cheia primară) a noii relații. Acest lucru este absolut normal. De asemenea, este perfect acceptabil să avem o relație în care toate atributele fac parte din identificatorul unic (adică nu există atribute care să nu facă parte din cheie).

5. Opțional, puteți înlocui cheia primară cu un singur atribut surogat pentru cheie. Dacă faceți acest lucru, trebuie să păstrați și atributele care compun cheia primară naturală, formată la pașii 2 și 4. Figura 1.7 prezintă rezultatul aducerii relației din figura 1.6 la prima formă normală.

Movie

MOVIE_ID (pk)	COPY_NUMBER (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT
1	1	Drama	Drama	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD

MOVIE_ID (pk)	COPY_NUMBER (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT
2	1	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD
2	2	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		VHS

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr
2	en
2	fr
2	es

Fig 1.7 Soluția primei forme normale

A doua formă normală: eliminarea dependențelor parțiale

Înainte de a explora a doua formă normală, trebuie să înțelegem conceptul *de dependență funcțională*.

Pentru această definiție, vom folosi două atribute arbitrare, inteligent denumite „A” și „B”. Atributul B este *dependent funcțional* de atributul A dacă în nici un moment nu există mai mult de o valoare a atributului B asociată cu o valoare dată a atributului A.

În relația Movie, putem să ne dăm seama cu ușurință că atributul Movie_Title este dependent funcțional de atributul Movie_ID, deoarece, în orice moment, poate exista o singură valoare Movie_Title pentru o valoare Movie_ID dată. Chiar faptul că valoarea Movie_ID definește în mod unic valoarea Movie_Title în relație înseamnă că Movie_Title este dependent funcțional de Movie ID.

Se spune că o relație este în *a doua formă normală* dacă îndeplinește următoarele criterii:

- Relația este în prima formă normală.
- Toate atributele non-cheie sunt dependente funcțional de identificatorul unic (cheia primară), *luat ca întreg*.

Aplicând aceste criterii relației Movie din fig. 1.7, este clar că avem câteva probleme. Identificatorul unic este o combinație a atributelor Movie_ID și Copy_Number. Totuși, numai atributele Date_Acquired, Date_Sold, Media_Format și Retail_Price depind de întregul identificator. Și este logic să fie așa. Indiferent câte copii ale unui film avem în baza de date, toate au aceleași valori pentru gen, categorie MPAA, titlu și an de producție. Cum a apărut această problemă? Ar trebui să fie clar că unele atribute descriu filmul în sine, în timp ce altele descriu, copiile pe care le deține (sau le-a deținut) magazinul din filmul respectiv. În esență, am amestecat atribute care descriu în aceeași relație două lucruri (entități) diferite (deși înrudite) din lumea reală. Nici nu e de mirare că am obținut un asemenea haos. A doua formă normală ne va ajuta să rezolvăm problemele.

A doua formă normală se aplică numai relațiilor care au identificatoare unice concatenate (adică formate din atribute multiple). Într-o relație care are un singur atribut ca identificator unic, este imposibil ca un alt atribut să depindă de o parte a identificatorului unic, deoarece acesta, fiind format dintr-un singur atribut, nu are părți componente. **Ca urmare, orice relație în prima formă normală care are cheia primară formată dintr-un singur atribut este automat în a doua formă normală.**

Movie

MOVIE_ID (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED
---------------	------------------	------------------	------------------	------------------	-------------	---------------

1	Drama	Drama	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003
2	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr
2	en
2	fr
2	es

Movie_Copy

MOVIE_ID (pk)	COPY_NUMBER (pk)	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT	RETAIL_PRICE
1	1	01/01/2005		DVD	19.96
2	1	10/01/2005		DVD	19.96
2	2	10/01/2005		VHS	15.95
3	1	10/01/2005	30/01/2005	DVD	29.99
3	2	15/02/2005		DVD	29.99

Fig. 1.8 Soluția pentru a doua formă normală

A treia formă normală: eliminarea dependențelor tranzitive

Pentru a înțelege a treia formă normală, trebuie să înțelegem mai întâi conceptul de dependență tranzitivă.

Despre un atribut care depinde de un atribut care nu este identificator unic (cheie primară) a relației se spune că *este dependent tranzitiv*.

Uitându-ne la relația Movie din figura 1.8, observăm că atributul Genre_Description depinde de atributul Genre_Code, iar MPAA_Rating_Description depinde de MPAA_Rating_Code. Pericolul păstrării acestor descrieri în relația Movie este faptul că, în final, cele două atribute ajung să depindă de înregistrarea unui film, ceea ce duce la toate cele trei anomalii de date prezentate mai devreme în acest capitol.

Se spune că o relație este în *a treia formă normală* dacă îndeplinește următoarele două criterii:

- Relația este în a doua formă normală.
- Nu există dependențe tranzitive (cu alte cuvinte, toate atributele non-cheie depind numai de identificatorul unic).

Pentru a aduce la a treia formă normală o relație aflată în a doua formă normală, mutăm atributele dependente tranzitiv în relații în care depind numai de cheia primară

Avem grijă să lăsăm atributul de care depind acestea în relația originală, cu rolul de cheie externă. Va trebui apoi să reconstruim vizualizarea originală printr-o uniune. Ca efect secundar, toate atributele ușor de calculat sunt eliminate ca încălcări ale criteriilor celei de-a treia forme normale.

De exemplu, într-o bază de date pentru vânzări, Suma Totală este obținută înmulțind Cantitatea Cumpărată cu Prețul Unitar; așa cum se observă cu ușurință, Suma Totală este dependentă de Cantitatea Cumpărată și de Prețul Unitar. Presupunând că toate cele trei atribute sunt dependente de identificatorul unic al relației care le conține, este ușor de văzut că Suma Totală (rezultatul calculat) este, de fapt, *dependentă tranzitiv* de celelalte două atribute. Figura 1.9 conține soluția în a treia formă normală.

Movie

MOVIE_ID (pk)	MOVIE_GENRE_CODE	MPAA_RATING_CODE	MOVIE_TITLE	RETAIL_PRICE_VHS	RETAIL_PRICE_DVD	YEAR_PRODUCED
1	Drama	R	Mystic River	58.97	19.96	2003
2	ActAd	R	The Last Samurai	15.95	19.96	2003

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr

1.2. CONCEPTE SQL

SQL a devenit limbajul universal pentru bazele de date relaționale și este acceptat de aproape toate sistemele RDBMS moderne. Fără îndoială, acceptarea pe scară largă este rezultatul timpului și eforturilor depuse pentru dezvoltarea caracteristicilor limbajului și a standardelor, crescând nivelul de portabilitate a codului SQL între diferitele produse RDBMS.

SQL (Structured Query Language - limbaj structurat de interogare) este un limbaj standard folosit pentru comunicarea cu bazele de date. Numele limbajului poate fi pronunțat pe litere (es-q-el) sau la fel ca și cuvântul englezesc „sequel”. O *interogare (query)* este o simplă cerere transmisă către baza de date, la care aceasta răspunde într-o anumită formă. SQL este limbajul folosit cel mai frecvent pentru interogarea bazelor de date. SQL este considerat un limbaj *neprocedural* sau *declarativ* ceea ce înseamnă că-i spuneți calculatorului ce rezultate vreți, fără să-i spuneți cum să le obțineți.

De exemplu, dacă vreți să obțineți media numerelor de pe o coloană, folosiți funcția AVG. Nu este nevoie să numărați valorile din coloană și să împărțiți suma acestora la numărul obținut - procesorul limbajului SQL din DBMS se ocupă de toate aceste lucruri în locul dumneavoastră.

Este important să înțelegem că SQL nu este un limbaj procedural, ca C, Pascal, Basic, FORTRAN, COBOL sau Ada. Un limbaj *procedural* folosește o serie de instrucțiuni executate secvențial. De asemenea, limbajele procedurale includ instrucțiuni care pot modifica secvența de execuție, prin ramificarea la alte porțiuni ale procedurii sau prin parcurgerea ciclică a unui set de instrucțiuni din procedură. Mulți producători de sisteme RDBMS oferă extensii procedurale ale limbajului SQL de bază, cum ar fi Oracle PL/SQL {Procedural Language/SQL} sau Microsoft Transact-SQL, dar rețineți că acestea sunt extensii SQL care formează noi limbaje - codul SQL pe care-l conțin rămâne neprocedural. De asemenea, SQL nu trebuie confundat cu limbajele orientate spre obiecte, precum Java și C++.

Simple spus, SQL este un limbaj pentru gestionarea și întreținerea bazelor de date relaționale, nu un limbaj potrivit pentru programarea generală a aplicațiilor, cum ar fi sistemele de prelucrare a comenzilor sau a plăților.

SQL este deseori folosit în combinație cu limbajele procedurale sau orientate spre obiecte menționate anterior pentru a manipula stocarea și extragerea datelor, folosind instrucțiuni din limbajul de programare cu destinație generală pentru alte sarcini de programare, precum prezentarea datelor pe o pagină web sau furnizarea răspunsurilor la informațiile introduse de utilizatori de la tastatură sau mouse. Atunci când este necesară interacționarea cu baza de date, instrucțiunile din limbajul procedural formează instrucțiunea SQL, o transmit către RDBMS în vederea prelucrării, primesc rezultatele returnate de RDBMS și le prelucrează într-un mod corespunzător.

1.2.1. Conectarea la baza de date

Atunci când folosiți limbajul SQL pe un calculator personal, cu o copie personală a unui sistem DBMS, precum Microsoft Access sau Oracle Personal Edition, toate componentele bazei de date rulează pe același sistem de calcul. Totuși, acest aranjament nu este potrivit pentru bazele de date care trebuie să fie folosite în comun de mai mulți utilizatori.

Ca urmare, sunt mult mai frecvent întâlnite situațiile în care baza de date este instalată într-o arhitectură client/server, ca în figura 2.1.

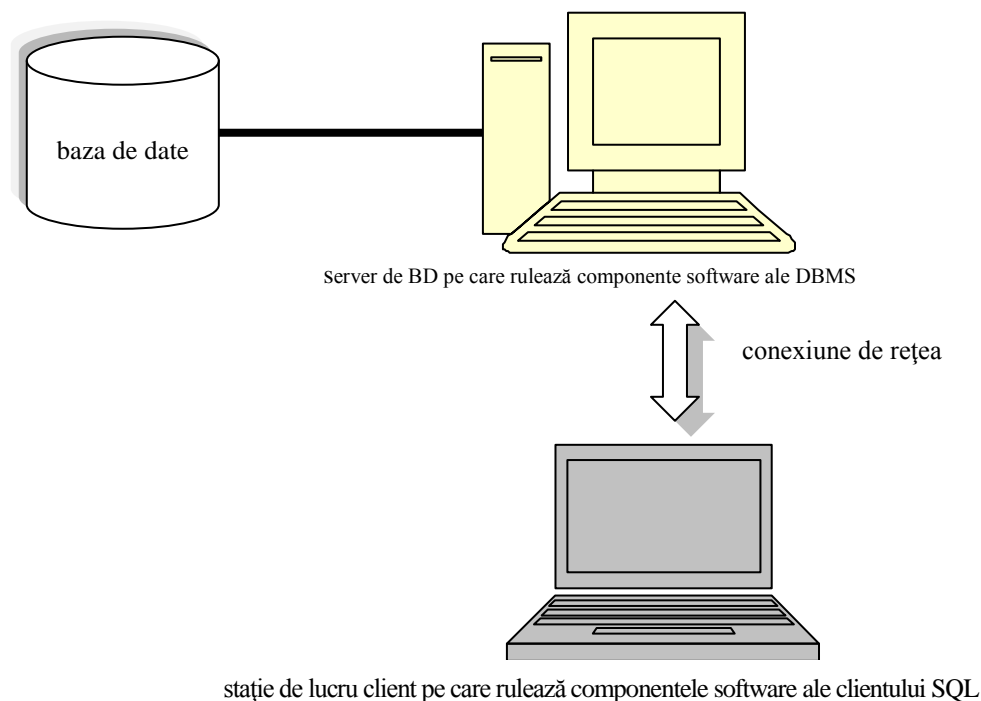


Fig. 2.1 Conexiunea clientului SQL cu baza de date

Într-o arhitectură client/server:

- Sistemul RDBMS rulează pe un *server*, care este un sistem de calcul partajat. Pentru scopurile acestei definiții un sistem mainframe poate fi considerat un server de dimensiuni mari.
- Fișierele care compun baza de date din punct de vedere fizic sunt stocate pe discuri conectate la serverul de baze de date.
- Utilizatorii care au acces la baza de date folosesc stații de lucru, numite *clienți*. Clientul trebuie să aibă o conexiune de rețea la baza de date, care poate fi o rețea privată, instalată acasă sau la birou, ori o rețea publică, precum Internet.
- Componentele software furnizate de producătorul DBMS rulează pe stațiile de lucru ale clienților pentru a oferi utilizatorilor posibilitatea să introducă instrucțiuni SQL, să le transmită sistemului DBMS în vederea prelucrării și să vadă rezultatele returnate de DBMS. În general, acest software se numește client SQL.

De reținut că: nimic nu ne oprește să instalăm clientul SQL pe același calculator cu sistemul DBMS. De fapt, mulți dezvoltatori care utilizează sisteme DBMS precum MySQL, Microsoft SQL Server și Oracle fac în mod obișnuit acest lucru, deoarece este foarte convenabil să aibă întregul mediu de dezvoltare pe un singur calculator, cum ar fi un laptop.

Totuși, în momentul în care este necesar accesul partajat al mai multor utilizatori, este mult mai convenabil și mai eficient să aveți o singură copie a sistemului DBMS pe un server partajat și să aveți numai clientul SQL instalat pe stația de lucru a fiecărui client.

În funcție de interfața cu utilizatorul de pe stația de lucru client, clienții SQL sunt clasificați în trei categorii:

- **în linia de comandă:** o interfață *în linia de comandă* se bazează exclusiv pe intrări și ieșiri de tip text, cu comenzile introduse de la tastatură și răspunsurile afișate ca mesaje de tip text. Principalul avantaj al interfețelor în linia de comandă este că pot fi rulate pe aproape orice sistem de operare.
- **grafici:** o interfață grafică cu utilizatorul (GUI - graphical user interface) rulează sub un tip oarecare de sistem bazat pe ferestre, cum ar fi X Window System, Mac OS sau Microsoft Windows, și afișează datele sau opțiunile comenzilor folosind elemente grafice, precum

pictograme, butoane și casete de dialog.

- **bazați pe web:** o interfață bazată pe web rulează pe serverul de baze de date, folosind un browser web de pe stația de lucru client pentru a interacționa cu utilizatorul bazei de date. Din punct de vedere tehnic, un client SQL bazat pe web nici nu este o aplicație client, deoarece nu există nici o componentă software specifică producătorului DBMS rulată pe stația de lucru a clientului. Totuși, aproape întotdeauna există componente furnizate de producătorul DBMS care sunt descărcate în fundal de browser-ul web pentru a asista în procesul de reprezentare grafică a formularelor web folosite pentru introducerea instrucțiunilor SQL și afișarea rezultatelor.

1.2.2. Convenții de sintaxă SQL

Această secțiune prezintă convențiile generale de sintaxă folosite pentru construirea instrucțiunilor SQL. Convențiile de sintaxă SQL sunt mai ușor de înțeles folosind un exemplu simplu. Instrucțiunea de mai jos returnează valorile `Movie_ID` și `Movie_Title` pentru toate filmele din magazinul de produse video pentru care categoria MPAA este „PG”:

```
SELECT MOVIE_ID, MOVIE_TITLE FROM MOVIE WHERE MPAA_RATING_CODE = 'PG';
```

Convențiile de bază sunt următoarele:

- Fiecare instrucțiune începe cu o comandă, de obicei sub forma unui singur cuvânt, care aproape întotdeauna este un verb (în limba engleză) care descrie o acțiune, în acest exemplu, instrucțiunea începe cu comanda `SELECT`.
- Fiecare instrucțiune se termină cu un delimitator, care este, de obicei, un caracter punct și virgulă (;).
- Instrucțiunile sunt construite într-o manieră similară cu propozițiile din limba engleză, cu unul sau mai multe spații pentru separarea elementelor de limbaj. Un *element de limbaj*, asemănător cu un cuvânt dintr-o propoziție, poate fi un cuvânt cheie (`SELECT`, `FROM`, `WHERE`), numele unui obiect al bazei de date (`MOVIE`, `MOVIE_ID`, `MOVIE_TITLE`), un operator (`=`) sau o constantă ('PG') care apare într-o instrucțiune.
- Instrucțiunile sunt scrise într-o formă liberă, ceea ce înseamnă că nu există reguli stricte privind poziția elementelor de limbaj pe o linie sau locul în care se poate face trecerea la o linie nouă. Totuși, în general nu este o idee bună să împărțiți un element de limbaj pe mai multe linii.
- Instrucțiunile sunt organizate într-o serie de clauze și, de obicei, clauzele trebuie să apară într-o anumită ordine atunci când sunt folosite (multe clauze sunt opționale). În exemplul nostru, există trei clauze, fiecare începând cu un cuvânt cheie (`SELECT`, `FROM`, `WHERE`).
- Elementele de limbaj SQL pot fi scrise cu litere mari, cu litere mici sau în combinații. Aceasta nu înseamnă că datele nu pot conține litere mici, ci că numele obiectelor din baza de date (tabele, coloane etc.) și comenzile trebuie să fie scrise cu litere mari.
- Virgulele sunt folosite pentru separarea articolelor dintr-o listă. În exemplul nostru, numele a două coloane sunt specificate într-o listă separată prin virgule (`MOVIE_ID`, `MOVIE_TITLE`). Spațiile care urmează după virgule sunt opționale puteți adăuga orice număr de spații, inclusiv zero.
- Șirurile de caractere care apar în instrucțiunile SQL trebuie să fie încadrate cu apostrofuri (unele implementări SQL permit și folosirea ghilimelelor). Constantele numerice nu sunt niciodată încadrate cu apostrofuri. Dacă în șirul de caractere trebuie să apară un caracter apostrof sunt inserate două apostrofuri unul lângă celălalt. De exemplu, dacă vreți să găsiți în baza de date un film numit *Sophie's Choice*, veți scrie clauza `WHERE` astfel;
 - `WHERE MOVIE_NAME = 'Sophie' 's Choice'`
- Numele obiectelor bazei de date sunt formate folosind *numai* litere, cifre și liniuțe de subliniere. Caracterul *underscore* (liniuța de subliniere) este folosit, de obicei, ca separator

între cuvinte, pentru îmbunătățirea lizibilității.

- În fiecare implementare SQL este definit un set *de cuvinte rezervate*, care sunt cuvinte cu o semnificație specifică pentru procesorul SQL al sistemului DBMS și, ca urmare, nu trebuie folosite într-un alt context — de exemplu ca nume pentru obiectele bazei de date.
- Un comentariu pe o singură linie începe cu două liniuțe de despărțire (--), Cele două liniuțe se pot afla la începutul unei linii, ceea ce înseamnă că întreaga linie este considerată comentariu, sau oriunde în cadrul liniei, caz în care restul liniei, până la sfârșit, este considerat comentariu. De exemplu:

--Acesta este un comentariu pe o singură linie în SQL.

- Un comentariu pe mai multe linii începe cu combinația dintre o diagonală la dreapta (slash) și un asterisc (/*) și continuă până la întâlnirea combinației inverse (*). Un exemplu de comentariu pe mai multe linii:

/ Acesta este un comentariu pe mai multe linii.*

*Continuă până la întâlnirea combinației de caractere care marchează sfârșitul comentariului. */*

Instrucțiunile SQL sunt împărțite în categorii, după funcțiile pe care le îndeplinesc. Categoriile de instrucțiuni, descrise în secțiunile următoare, sunt:

- Limbajul de definire a datelor (DDL - Data Definition Language)
- Limbajul de interogare a datelor (DQL - Data Query Language)
- Limbajul de manipulare a datelor (DML - Data Manipulation Language)
- Limbajul pentru controlul datelor (DCL - Data Control Language)
- Comenzile pentru controlul tranzacțiilor (Transaction Control Commands)

Limbajul de definire a datelor (DDL): include instrucțiuni SQL care permit utilizatorului bazei de date să creeze și să modifice structura obiectelor bazei de date, cum ar fi tabele, vizualizări și indexuri. Instrucțiunile SQL care folosesc comenzile CREATE, ALTER și DROP sunt considerate parte a DDL.

Este important să înțelegem că instrucțiunile DDL afectează containerele care stochează datele în baza de date, nu datele propriu-zise. Ca urmare, există instrucțiuni DDL pentru crearea, ștergerea și modificarea tabelor, dar nici una dintre aceste instrucțiuni nu oferă posibilitatea de a crea sau modifica rânduri de date din tabelele respective.

Limbajul de interogare a datelor (DQL): include instrucțiuni SQL care permit obținerea datelor din baza de date. Deși reprezintă o parte foarte importantă a limbajului SQL, DQL este format din instrucțiuni care folosesc o singură comandă: SELECT.

Limbajul de manipulare a datelor (DML): include instrucțiuni SQL care permit utilizatorului bazei de date să adauge date în baza de date (sub forma rândurilor din tabele), să șteargă date și să modifice datele existente în baza de date. Instrucțiunile SQL care folosesc comenzile INSERT, UPDATE și DELETE sunt considerate parte a DML.

Limbajul pentru controlul datelor (DCL): include instrucțiuni SQL care permit administratorilor să controleze accesul la datele din baza de date și folosirea diferitelor privilegii ale sistemului DBMS, cum ar fi privilegiul de oprire și pornire a bazei de date. Instrucțiunile SQL care folosesc comenzile GRANT și REVOKE sunt considerate parte a DCL.

Comenzile pentru controlul tranzacțiilor: o *tranzacție în baza de date* este un set de comenzi pe care utilizatorul bazei de date vrea să le trateze ca pe o unitate funcțională de tip 'totul sau nimic', înțelegând prin aceasta că întreaga tranzacție trebuie să reușească sau să eșueze. Comenzile pentru controlul tranzacțiilor (Transaction Control Commands) nu respectă cu exactitate sintaxa instrucțiunilor SQL, dar afectează puternic comportamentul instrucțiunilor SQL incluse în tranzacții.

1.3. DEFINIREA ȘI INTEROGAREA DATELOR FOLOSIND LIMBAJUL SQL

1.3.1. Instrucțiuni DDL (Data Definition Language)

Instrucțiunile DDL (Data Definition Language) definesc obiectele bazei de date, dar nu inserează și nu actualizează date în obiectele respective (această funcție fiind îndeplinită de instrucțiunile DML).

În SQL, există trei comenzi de bază pentru instrucțiunile DDL:

- **CREATE** - Creează în baza de date un nou obiect, de tipul specificat în instrucțiune. Deoarece sintaxa este diferită, vom prezenta separat instrucțiunile CREATE DATABASE, CREATE TABLE, CREATE INDEX și CREATE VIEW.
- **ALTER** - Modifică definiția unui obiect existent în baza de date, de tipul specificat în instrucțiune.
- **DROP** - Șterge (distruge) un obiect existent în baza de date, de tipul specificat în instrucțiune.

Instrucțiunea CREATE DATABASE

Sintaxa generală pentru instrucțiunea CREATE DATABASE este

```
CREATE DATABASE nume_baza_de_date [opțiuni specifice producătorului]
```

Instrucțiunea CREATE TABLE

CREATE TABLE este una din instrucțiunile fundamentale din SQL. Modelul relațional cere ca toate datele stocate să fie ancorate într-un tabel, așa că posibilitatea de a stoca orice într-o bază de date începe întotdeauna cu crearea unui tabel. Sintaxa de bază pentru instrucțiunea CREATE TABLE este

```
CREATE TABLE nume_tabel ( <definiție_coloana> [,<definiție_coloană> ...])  
[, <restricție_tabel> ... ] ;
```

Fiecare instrucțiune include numele tabelului și o listă cu una sau mai multe definiții de coloane, separate prin virgule și încadrate între paranteze. Numele tabelului trebuie să fie unic în baza de. Fiecare tabel trebuie să aibă cel puțin o coloană.

Definirea coloanelor în SQL DDL

Sintaxa de bază folosită pentru definirea coloanelor unui tabel este:

```
<definiție_coloana> :  
nume_coloană tip_de_date [DEFAULT expresie] [ NULL | NOT NULL]  
[<restricție_coloană>]
```

Restricțiile coloanelor

Restricțiile unei coloane limitează (constrâng) într-un mod oarecare valorile ce pot fi stocate într-o coloană a unui tabel.

Din punct de vedere tehnic, clauzele DEFAULT și NULL sau NOTNULL sunt forme speciale de restricții, dar acestea nu sunt implementate întotdeauna în același mod în toate sistemele DBMS. Restricția unei coloane se poate referi la o singură coloană, dar există o modalitate simplă de ocolire, deoarece orice restricție de coloană poate fi rescrisă ca restricție de tabel. Restricțiile coloanelor pot avea oricare dintre următoarele forme:

Clauza DEFAULT

O expresie care este aplicată coloanei atunci când în tabel este inserat un nou rând, care nu conține o valoare explicită pentru coloana respectivă. Expresia poate fi orice expresie validă care poate fi interpretată de SQL, cum ar fi o constantă, o funcție SQL sau o altă sintaxă care, în urma evaluării de către motorul SQL din DBMS, produce o valoare corespunzătoare pentru coloana respectivă.

Ca *exemplu*, observați clauza DEFAULT 'N' pentru coloana CUSTOMER_HOLD_INDIC din tabelul CUSTOMER_ACCOUNT. Prin specificarea acestei valori prestabilite vă asigurați că orice cont de client nou creat va primi întotdeauna valoarea 'N' (contul nu este blocat) dacă instrucțiunea care inserează noul rând nu furnizează o valoare pentru această coloană sau folosește cuvântul cheie DEFAULT pentru valoarea coloanei. O altă

utilizare obișnuită a clauzei DEFAULT este inserarea unor date, cum ar fi stocarea datei curente pentru efectuarea unei tranzacții, la inserarea noilor rânduri în baza de date. Iată sintaxa SQL și un exemplu de coloană cu o clauză DEFAULT:

```
[DEFAULT expresie]
```

Exemplu:

```
CUSTOMER_HOLD_INDIC CHAR(1) DEFAULT 'N' NOT NULL
```

Restricția **NULL | NOT NULL**

Specificarea cuvântului cheie NULL permite stocarea valorilor nule într-o coloană, în timp ce NOT NULL nu permite stocarea valorilor nule în coloana respectivă. Iată sintaxa SQL și câteva exemple:

```
NULL | NOT NULL
```

Exemple:

```
DATE_ENROLLED DATE NOT NULL
```

```
DATE_TERMINATED DATE NULL
```

Restricția **CHECK**

O restricție de verificare (check) poate fi folosită pentru impunerea unei reguli care poate fi aplicată unei singure coloane a unui tabel. Condiția inclusă în restricție trebuie să fie îndeplinită ori de câte ori datele din coloana respectivă a tabelului sunt modificate - în caz contrar, sistemul DBMS va respinge modificarea și va afișa un mesaj de eroare. Condiția din restricția CHECK a unei coloane nu poate referi nici o altă coloană a tabelului.

Iată sintaxa restricției CHECK și un exemplu:

```
[CONSTRAINT nume_restricție] CHECK (conditie)
```

Exemplu:

```
CREDIT_CARD_ON_FILE_INDIC CHAR(1) NOT NULL
```

```
CHECK (CREDIT_CARD_ON_FILE_INDIC IN 'Y', 'N')
```

Restricția **UNIQUE**

O restricție UNIQUE impusă asupra unei coloane garantează unicitatea valorilor din coloana respectivă a tabelului, de obicei cu ajutorul unui index creat automat de DBMS.

Iată sintaxa restricției de unicitate și un exemplu de utilizare:

```
[CONSTRAINT nume_restricție] UNIQUE
```

Exemplu:

```
CUSTOMER_ACCOUNT_ID INTEGER NOT NULL UNIQUE
```

Restricția **PRIMARY KEY**

O restricție de cheie primară (PRIMARY KEY) impusă asupra unei coloane declară coloana respectivă ca fiind cheia primară a tabelului, ceea ce înseamnă că în coloana respectivă nu pot exista valori nule, iar valorile trebuie să fie unice în cadrul tabelului. Ca și în cazul restricțiilor de unicitate, majoritatea sistemelor DBMS impun restricția de cheie primară cu ajutorul unui index creat automat.

Iată sintaxa restricției de cheie primară și un exemplu de utilizare:

```
[CONSTRAINT nume_restricție] PRIMARY KEY
```

Exemplu:

```
CUSTOMER_ACCOUNT_ID INTEGER NOT NULL PRIMARY KEY
```

Restricția referențială (**FOREIGN KEY**)

O restricție referențială impusă asupra unei coloane (numită uneori și restricție de cheie externă) definește relația dintre o cheie externă și o cheie primară, astfel încât sistemul DBMS să poată garanta că valoarea, dacă nu este nulă, referă întotdeauna valoarea unei chei primare existente. Iată sintaxa restricției referențiale:

```
[CONSTRAINT nume_restricție]
```

```
REFERENCES nume_tabel (nume_coloană)
```

```
[ON DELETE CASCADE | ON DELETE SET NULL]
```

Exemplu:

```
MPAA_RATING_CODE CHAR(5) NOT NULL
REFERENCES MPAA_RATING (MPAA_RATING_CODE)
```

Clauza opțională ON DELETE spune sistemului DBMS ce să facă atunci când este șters rândul referit din tabelul părinte (rândul care conține cheia primară corespondentă), cu opțiunea de a șterge toate rândurile care conțin cheia externă (CASCADE) sau de a insera valori nule pentru toate cheile externe (SET NULL). Rețineți că majoritatea sistemelor DBMS, dar nu toate, acceptă clauza ON DELETE.

Restricțiile tabelor

Restricția unei coloane poate fi rescrisă și ca restricție a întregului tabel, astfel încât clauza care definește restricția să apară în instrucțiunea CREATE TABLE după definițiile tuturor coloanelor, nu după definiția unei coloane. Principalul avantaj al restricțiilor la nivelul tabelului este că pot referi mai multe coloane. Semnificația fiecărei restricții a fost deja discutată (în secțiunea despre restricțiile coloanelor) așa că aici vom prezenta numai sintaxa generală și câteva exemple.

Toate exemplele folosesc tabelul CUSTOMER_ACCOUNT, dar unele au fost modificate, astfel încât să ilustreze ideile principale prezentate.

Restricția CHECK

```
[CONSTRAINT nume_restricție] CHECK (condiție)
```

Exemplu:

```
CONSTRAINT CK_CUSTOMER_DEPOSIT_AMOUNT
CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR
CUSTOMER_DEPOSIT_AMOUNT
IS NULL)
```

Restricția din exemplul de mai sus împiedică stocarea unei valori negative în coloana CUSTOMER_DEPOSIT_AMOUNT. Observați operatorul SAU, folosit pentru a permite stocarea valorilor nule în coloană. Dacă nu ar fi fost inclusă această condiție, coloana nu ar fi acceptat valori nule, deoarece o valoare nulă nu este mai mare sau egală cu zero.

Restricția UNIQUE

```
[CONSTRAINT nume_restricție] UNIQUE(nume_coloana[, nume_coloana ...,])
```

Exemplu:

```
CONSTRAINT UK_CUST_ACCT_DATE_ENROLLED
UNIQUE (CUSTOMER_ACCOUNT_ID, DATE_ENROLLED)
```

Conform acestei restricții, combinația de coloane CUSTOMER_ACCOUNT_ID și DATE_ENROLLED trebuie să fie unică în rândurile din tabelul CUSTOMER_ACCOUNT.

În acest exemplu, coloana CUSTOMER_ACCOUNT_ID este oricum unică, așa că impunerea noii restricții nu prea are sens, dar am inclus-o aici pentru a ilustra folosirea unei restricții de unicitate bazate pe mai multe coloane.

Restricția PRIMARY KEY

```
[CONSTRAINT nume_restricție] PRIMARY KEY(nume_coloana[,nume_coloana...,])
```

Exemplu:

```
CONSTRAINT PK_CUSTOMER_ACCOUNT
PRIMARY KEY (CUSTOMER_ACCOUNT_ID)
```

Restricția de mai sus este chiar definiția cheii primare din tabelul CUSTOMER_ACCOUNT folosit ca exemplu în această secțiune, dar în acest caz am denumit explicit restricția.

Restricția referențială (FOREIGN KEY)

```
[CONSTRAINT nume_restricție]
FOREIGN KEY (nume_coloană [, nume_coloană..., ]) REFERENCES
nume_tabel(nume_coloană [, nume_coloană... ]) [ON DELETE CASCADE | ON DELETE
SET NULL)
```


Observați că, spre deosebire de forma pentru coloane a restricției referențiale, aceasta poate referi coloane multiple. Așa cum a fost proiectat, tabelul CUSTOMER_ACCOUNT nu are coloane chei externe, dar să ne gândim la un alt model.

Unii proiectanți de baze de date nu sunt de acord cu folosirea restricțiilor CHECK pentru a controla valorile din coloane, deoarece adăugarea sau eliminarea valorilor implică modificarea proiectului bazei de date. Să presupunem, de exemplu, că la o îmbunătățire ulterioară a bazei de date pentru magazinul de produse video trebuie să permitem stocarea unei noi valori, 'E' (de la „exceptat”), în coloana CREDIT_CARD_ON_FILE_INDIC. Puteți modifica restricția CHECK pentru a permite stocarea noii valori în tabel, dar dacă ați fi pus toate valorile și descrierile într-un tabel separat (deseori numit tabel „de coduri”, „referință” sau „de căutare”), ar fi fost suficient să adăugați noul cod în tabelul respectiv, fără nici o modificare asupra proiectului bazei de date. Tocmai pentru obținerea acestei flexibilități, baza de date a magazinului de produse video conține tabele pentru informații precum MPAA_RATING_CODE - dacă asociația MPAA își schimbă sistemul de clasificare, este suficient să modificați datele din tabelul de coduri. De asemenea, tabelele de coduri sunt o sursă elegantă pentru popularea cu valori a listelor derulante din componente de aplicație, precum formularele din paginile web.

Presupunând că a fost creat un tabel numit CARD_ON_FILE_TYPE, cu o cheie primară numită CARD_ON_FILE_CODE, iată cum arată restricția referențială care definește coloana CREDIT_CARD_ON_FILE_INDIC ca fiind cheie externă:

```
CONSTRAINT          FK_CARD_ON_FILE_INDIC          FOREIGN          KEY
(CREDIT_CARD_ON_FILE_INDIC)          REFERENCES          CARD_ON_FILE_TYPE
(CARD_ON_FILE_CODE)
```

Așa cum puteți vedea, nu întotdeauna există un singur mod „corect” de proiectare a unei baze de date, ci mai multe soluții din care puteți alege. Altfel spus, proiectarea bazelor de date nu este o știință exactă. În general, este recomandabil să dați cheii externe același nume ca și cheii primare, dar așa cum puteți vedea din acest exemplu, SQL vă permite să folosiți un alt nume, dacă așa vreți sau așa trebuie.

Instrucțiunea CREATE INDEX

CREATE [UNIQUE] INDEX *nume_index* ON *nume_tabel* (*nume_coloană* [*nume_coloană* [ASC | DESC]...]);

- Cuvântul cheie opțional UNIQUE definește indexul ca unic, însemnând că nu pot exista două rânduri din tabel cu exact aceeași combinație de valori în coloanele specificate.
- Cuvântul cheie opțional ASC creează indexul în ordine crescătoare, în timp ce DESC creează indexul în ordine descrescătoare. Dacă nu este specificată nici una dintre cele două opțiuni, ordinea prestabilită este crescătoare.
- Un index trebuie să aibă cel puțin o coloană, dar, practic, nu există o limită superioară a numărului de coloane.

Indexurile sunt instrumente puternice, deoarece permit sistemului DBMS să găsească datele mult mai repede, așa cum indexul unei cărți vă permite să găsiți rapid ceea ce vă interesează. Mai mult, indexurile pe coloanele cheilor externe cresc mult performanțele la unirea tabelor. Totuși, indexurile ocupă spațiu de stocare și trebuie să fie întreținute de fiecare dată când se modifică o valoare de pe o coloană referită de un index, trebuie să fie modificat și indexul corespunzător. Sistemul DBMS întreține automat indexul, dar activitatea de întreținere consumă din resursele calculatorului.

Instrucțiunea CREATE VIEW

Vizualizările oferă mari avantaje utilizatorilor unei baze de date, deoarece permit ajustarea datelor afișate în funcție de cerințele individuale și maschează operațiile complexe. Atunci când sunt create corect, vizualizările implică suprasarcini minime și nu stochează date.

În esență, o vizualizare este o interogare SQL stocată, care poate fi referită de instrucțiunile SQL DML și DQL ca și cum ar fi un tabel real. Unii consideră că vizualizările sunt „tabele virtuale”, deoarece se comportă la fel ca tabelele (cu unele restricții), dar nu există ca tabele fizice.

Sintaxa generală a instrucțiunii CREATE VIEW este:

CREATE [OR REPLACE] VIEW *nume_vizualizare* AS *interogare_sql*

- Cuvântul cheie opțional OR REPLACE elimină necesitatea de a șterge o vizualizare existentă înainte de a o crea din nou. Dacă specificați parametrul opțional OR REPLACE și vizualizarea există deja, este înlocuită, iar dacă nu există, noua vizualizare este adăugată în baza de date.
- Numele vizualizării trebuie să respecte aceleași reguli de denumire ca și tabelele și alte obiecte ale bazei de date. În interogările SQL sunt specificate numele obiectelor din care sunt selectate datele, dar nu și tipul acestor obiecte. Ca urmare, numele vizualizărilor trebuie să fie unice pentru toate tabelele și vizualizările din baza de date. Cu alte cuvinte, numele vizualizărilor și numele tabelor trebuie să provină din același *spațiu de nume*, adică același domeniu de nume.
- Interogarea SQL inclusă în definiția vizualizării poate fi orice instrucțiunea SQL SELECT validă. Vom învăța despre această instrucțiune SQL esențială în capitolul 4. Crearea vizualizărilor urmează o cale naturală de evoluție - lucrați cu interogarea SQL, faceți modificările necesare și rulați din nou interogarea până când obțineți rezultatele dorite. Apoi adăugați instrucțiunea CREATE VIEW în fața interogării cu care ați lucrat și rulați instrucțiunea pentru a stoca permanent interogarea în baza de date, ca vizualizare. Aceasta este o modalitate foarte productivă (și plăcută) de a lucra cu bazele de date.

Instrucțiunea ALTER TABLE

După ce creați un tabel, aproape orice ați specificat în instrucțiunea CREATE TABLE poate fi modificat folosind instrucțiunea ALTER TABLE.

Utilizarea instrucțiunii ALTER TABLE este un alt domeniu în care au un rol important stilul și preferințele personale. Mulți administratori de baze de date preferă să folosească instrucțiuni CREATE TABLE cât mai simple, evitând să definească restricții în instrucțiunile CREATE TABLE. Aceștia adaugă după instrucțiunea CREATE TABLE instrucțiuni ALTER TABLE prin care specifică toate restricțiile necesare (cheie primară, cheie externă, unicitate, verificare și așa mai departe). Dezavantajul acestei metode este acela că necesita scrierea unei cantități mai mari de cod. Pe de altă parte, instrucțiunea CREATE TABLE este mult mai ușor de înțeles fără restricții, iar scrierea separată a restricțiilor simplifică refolosirea instrucțiunilor, dacă este nevoie să eliminați și apoi să creați din nou restricțiile.

***Adăugarea unei coloane la un tabel.** Definirea coloanei se face cu aceeași sintaxă ca și în cazul instrucțiunii CREATE TABLE.

```
ALTER TABLE nume_tabel ADD ( <definiție_coloană> [,<definiție_coloană>...]);
```

Exemplu:

```
ALTER TABLE CUSTOMER_ACCOUNT ADD (CUSTOMER_HOLD_DATE DATE  
NULL,
```

```
HOLD_PLACED_BY VARCHAR(50));
```

***Modificarea definiției unei coloane.**

```
ALTER TABLE nume_tabel MODIFY | CHANGE [COLUMN] ( <definiție_coloană>  
[,<definiție_coloană>...]);
```

Exemplu:

```
ALTER TABLE CUSTOMER_ACCOUNT MODIFY  
(CUSTOMER_DEPOSIT_AMOUNT NUMERIC(7,2) DEFAULT 0 NOT NULL);
```

* **Adăugarea unei restricții.** Definiția restricției este identică cu definiția unei restricții care ar putea apărea într-o instrucțiune CREATE TABLE.

```
ALTER TABLE nume_tabel ADD CONSTRAINT <definiție_restricție>;
```

Exemplu:

```
ALTER TABLE CUSTOMER_ACCOUNT ADD CONSTRAINT CK_CUSTOMER  
DEPOSIT_AMOUNT CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR  
CUSTOMER_DEPOSIT_AMOUNT IS NULL);
```

* **Ștergerea cheii primare a unui tabel.** Dacă cheia primară este referită de restricții referențiale, trebuie mai întâi șterse restricțiile respective.

```
ALTER TABLE nume_tabel DROP PRIMARY KEY;
```

* **Redenumirea unei coloane.**

```
ALTER TABLE nume_tabel RENAME nume_vechi_coloană TO nume_nou_coloană;
```

În MySQL este implementată varianta:

```
ALTER TABLE nume_tabel  
    CHANGE [COLUMN] nume_col_vechi nume_col_noua column_definition  
    [FIRST|AFTER col_name]
```

Instrucțiunea DROP

Instrucțiunea DROP este cea mai simplă dintre instrucțiunile DDL. Sintaxa de bază este:

```
DROP <tip_obiect> nume_obiect [<opțiuni_de_ștergere>]
```

- Tipul de obiect specifică tipul obiectului care urmează să fie șters, cum ar fi INDEX, TABLE sau VIEW.
- Opțiunile de ștergere sunt specifice fiecărui DBMS. În general, dacă un tabel este referit de o restricție referențială, sistemul DBMS nu vă va permite să ștergeți tabelul

Iată câteva exemple:

```
DROP TABLE CUSTOMER_ACCOUNT;  
DROP TABLE CUSTOMER_ACCOUNT CASCADE CONSTRAINTS; (Oracle)  
DROP TABLE CUSTOMER_ACCOUNT CASCADE; (MySQL / PostgreSQL)  
DROP INDEX IX_MOVIE_TITLE ON MOVIE;
```

1.3.2. Instrucțiuni DQL (Data Query Language)

Limbajul SQL de interogare a datelor (DQL - Data Query Language) include o singură comandă, dar una foarte importantă: SELECT. Comanda SELECT este folosită pentru a obține date din baza de date (fără să le modifice), astfel încât acestea să poate fi prelucrate de o aplicație sau afișate pentru un utilizator.

Rezultatul unei instrucțiuni SELECT, numit *set de rezultate*, este returnat întotdeauna sub forma unui tabel (adică rânduri și coloane). Nu uitați că SQL este un limbaj neprocedural, așa că specificați rezultatele pe care vreți să le obțineți (adică modul în care vreți să fie returnat setul de rezultate), nu și modul în care vor fi obținute acestea.

Instrucțiunea SELECT de bază

Forma elementară a instrucțiunii SELECT conține două clauze:

- **SELECT [DISTINCT]** - Specifică lista de coloane care urmează să fie returnate în setul de rezultate, separate prin virgule. Puteți folosi simbolul asterisc (*) în locul listei de coloane pentru a selecta toate coloanele dintr-un tabel sau dintr-o vizualizare. Cuvântul cheie DISTINCT poate fi adăugat după cuvântul cheie SELECT pentru a elimina rândurile duplicate din rezultatele interogării.
- **FROM** - Specifică lista tabelelor sau vizualizărilor din care urmează să fie selectate datele. În locul numelor reale ale tabelelor sau vizualizărilor puteți folosi *sinonime*, adică pseudonime pentru tabele sau vizualizări definite în baza de date.

Exemplul următor selectează coloanele MOVIE_GENRE_CODE, MPAA_RATING_CODE și MOVIE_TITLE din tabelul MOVIE.

```
SELECT MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE FROM  
MOVIE;
```

Pseudonime pentru numele coloanelor

Se observă din interogarea anterioară că numele coloanelor din tabel apar automat ca titluri ale coloanelor din setul de rezultate. Totuși, nu este obligatoriu să se întâmple așa, deoarece instrucțiunea SQL vă pune la dispoziție o modalitate simplă de a specifica pseudonime pentru coloane. Pseudonimele (*aliases*) specificate devin numele coloanelor din setul de rezultate.

Atenție la un aspect - pseudonimele nu există decât după rularea instrucțiunii SQL, așa că nu pot fi folosite în alte părți ale instrucțiunii SQL. Pseudonimul unei coloane este specificat prin plasarea cuvântului cheie „AS” după numele coloanei în lista SELECT (cu cel puțin un spațiu înainte și după), urmat de numele pe care vrei să-l atribuieți coloanei în setul de rezultate. Iată cum arată instrucțiunea SQL rulată mai devreme, dar cu coloana MOVIE_GENRE_CODE redenumită GENRE și coloana MPAA_RATING_CODE redenumită RATING.

```
SELECT MOVIE_GENRE_CODE AS GENRE,
MPAA_RATING_CODE AS RATING, MOVIE_TITLE FROM MOVIE;
```

Sortarea rezultatelor

Rezultatele interogărilor sunt deseori mult mai utile dacă specificați pentru rândurile returnate o ordine care să aibă o semnificație pentru persoana sau aplicația care folosește informațiile. Nu există nici o garanție în privința ordinii în care sunt returnate rândurile din setul de rezultate *decât dacă* ordinea dorită este specificată în interogare.

În SQL, acest lucru este făcut prin adăugarea în instrucțiunea SELECT a clauzei **ORDER BY**, cu o listă de una sau mai multe coloane care vor fi folosite pentru sortarea rândurilor în ordine ascendentă sau descendentă, în conformitate cu valorile datelor din coloane.

În cazul instrucțiunii SELECT folosite mai devreme, să presupunem că ar fi utilă ordonarea ascendentă a rândurilor după coloanele MPAA_Rating și Movie_Genre_Code. Din perspectiva umană, cel mai bine este să plasați aceste coloane pe primele poziții în rezultatele interogării și să le specificați în aceeași ordinea în lista ORDER BY (cel puțin în limbile citite de la stânga la dreapta). Astfel, ordonarea rândurilor este evidentă pentru cel care citește rezultatele. Mai jos este prezentată instrucțiunea SELECT modificată.

```
SELECT MPAA_RATING_CODE AS RATING, MOVIE_GENRE_CODE AS GENRE,
MOVIE_TITLE FROM MOVIE ORDER BY MPAA_RATING_CODE,
MOVIE_GENRE_CODE;
```

Utilizarea clauzei WHERE pentru filtrarea rezultatelor

SQL folosește clauza "WHERE pentru a filtra rândurile ce urmează să fie afișate. Așa cum ați văzut deja, o interogare fără o clauză WHERE returnează un set de rezultate care conține toate rândurile din tabelele sau vizualizările referite în clauza FROM. Dacă este inclusă o clauză WHERE, sunt folosite regulile algebrei booleene, numite astfel după numele logicianului George Boole, evaluând clauza WHERE pentru fiecare rând de date. În rezultatele interogării sunt afișate numai rândurile pentru care clauza WHERE este evaluată la valoarea logică „adevărat”.

Operatori de comparare

Operatorii de comparare sunt folosiți în clauza WHERE pentru compararea a două valori, având ca rezultat o valoare logică de „adevărat” sau „fals”. Cele două valori comparate pot fi constante furnizate în clauza WHERE, valori ale unor coloane din baza de date sau combinații ale celor două. Operatorii de comparare care pot fi folosiți în clauza WHERE sunt prezentați în tabelul următor:

O	Descriere
=	Egal cu
<	Mai mic decât
<=	Mai mic sau egal
>	Mai mare decât
>=	Mai mare sau egal
!=	Diferit de
<>	Diferit de (standard)

Exemple:

- Afișați toate filmele pentru care MPAA_Rating are valoarea PG-13

```
SELECT MPAA_RATING_CODE AS RATING, MOVIE_TITLE
FROM MOVIE WHERE MPAA_RATING_CODE = 'PG-13'
ORDER BY MOVIE_TITLE;
```
- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (DVD Retail

Price) mai mic de 19.99, în ordinea descrescătoare a prețurilor.

```
SELECT RETAIL_PRICE_DVD, MOVIE_TITLE  
FROM MOVIE WHERE RETAIL_PRICE_DVD < 19.99  
ORDER BY RETAIL_PRICE_DVD DESC;
```

Operatori conjunctivi

Uneori sunt necesare condiții multiple pentru a îngusta setul de rezultate al unei interogări. Atunci când sunt folosite mai multe condiții, ele trebuie să fie combinate din punct de vedere logic în clauza WHERE, iar aceasta este sarcina *operatorilor conjunctivi*. Acești operatori sunt:

- **AND (ȘI)** Clauza WHERE este evaluată ca „adevărată” dacă *toate* condițiile conectate cu operatorul AND sunt adevărate.
- **OR (SAU)** Clauza WHERE este evaluată ca „adevărată” dacă *oricare* din condițiile conectate cu operatorul OR este adevărată.

Lucrurile devin complicate dacă operatorii AND și OR sunt combinați în aceeași clauză WHERE. Operatorul AND are prioritate mai mare și, ca urmare, este evaluat înaintea operatorilor OR. Condițiile din interiorul parantezelor sunt evaluate întotdeauna primele.

Iată un *exemplu* de folosire a operatorilor conjunctivi;

- Afișați toate filmele pentru care categoria MPAA este PG-13 și prețul de vânzare cu amănuntul pentru formatul DVD este 19.99 sau mai mic, în ordinea crescătoare a prețurilor.
SELECT MPAA_RATING_CODE AS RATING, RETAIL_PRICE_DVD AS PRICE,
MOVIE_TITLE FROM MOVIE WHERE MPAA_RATING_CODE = 'PG-13'
AND RETAIL_PRICE_DVD <= 19.99 ORDER BY RETAIL_PRICE_DVD;

Operatori logici

Operatorii logici folosesc cuvinte cheie în locul simbolurilor la formarea expresiilor de comparare. Operatorii logici disponibili în majoritatea implementărilor SQL sunt prezentați în secțiunile care urmează. La oricare dintre acești operatori poate fi adăugat cuvântul cheie NOT, pentru a inversa valoarea logică a comparației.

IS NULL

Operatorul IS NULL este folosit pentru a determina dacă o valoare este nulă. Este important să rețineți că valorile nule din baza de date nu sunt egale cu nimic altceva, nici chiar cu alte valori nule. Din această cauză, o condiție precum "= NULL" este întotdeauna incorectă. Niciodată nimic nu este egal cu o valoare nulă.

Exemplu:

- Găsiți toate conturile de clienți active, adică toate conturile pentru care coloana DATE_TERMINATED conține o valoare nulă:
SELECT CUSTOMER_ACCOUNT_ID FROM CUSTOMER_ACCOUNT
WHERE DATE_TERMINATED IS NULL;
- Găsiți toate conturile de clienți inactive, adică toate conturile pentru care coloana DATE_TERMINATED conține o altă valoare decât NULL:
SELECT CUSTOMER_ACCOUNT_ID FROM CUSTOMER_ACCOUNT
WHERE DATE_TERMINATED IS NOT NULL;

BETWEEN

Operatorul BETWEEN este folosit pentru a determina dacă o valoare se încadrează într-un interval specificat. Intervalul este specificat folosind o valoare minimă și o valoare maximă, fiind un interval *inclusiv*, ceea ce înseamnă că include și valorile specificate. Aceasta este o prescurtare elegantă, ușor de citit și de înțeles, pentru specificare unei condiții de interval. De exemplu, condiția „WHERE MOVIE_ID BETWEEN 7 AND 9” este aceeași cu condiția „WHERE MOVIE_ID >= 7 AND MOVIE_ID <= 9.

Exemplu:

- Afișați toate filmele cu prețul de vânzare cu amănuntul pentru formatul DVD (coloana RETAIL_PRICE_DVD) între 14.99 și 19.99, ordonate crescător după preț. Observați că în setul de rezultate sunt incluse și filmele pentru care prețul este exact 14.99 sau 19.99.

```
SELECT MOVIE_TITLE, RETAIL_PRICE_DVD
FROM MOVIE WHERE RETAIL_PRICE_DVD BETWEEN 14.99 AND 19.99
ORDER BY RETAIL_PRICE_DVD;
LIKE
```

Operatorul LIKE este folosit pentru a compara o valoare de tip caracter cu un tipar, returnând valoarea logică „adevărat” dacă valoarea de tip caracter se încadrează în tipar și „fals” în caz contrar. Pentru definirea tiparului pot fi folosite două caractere de înlocuire:

- Liniuța de subliniere (). Caracterul liniuța de subliniere poate fi folosit drept caracter de înlocuire pozițional, ceea ce înseamnă că se potrivește cu orice caracter aflat pe poziția respectivă în șirul de caractere evaluat.
- Procent (%). Simbolul procent (%) poate fi folosit drept caracter de înlocuire nepozițional, ceea ce înseamnă că se potrivește cu orice număr de caractere, indiferent de lungime.

Exemplu de utilizare a operatorului LIKE:

```
SELECT 'David!' LIKE 'David_';    // returnează valoarea 1
SELECT 'David!' LIKE 'David\_' ;  // returnează valoarea 0
```

- Afișați toate titlurile de filme care conțin șirul de caractere „on”:

```
SELECT MOVIE_TITLE FROM MOVIE WHERE MOVIE_TITLE LIKE '%on%';
```

sau

```
SELECT 'David!' LIKE '%D%v%';    // returnează valoarea 1
```

IN

Operatorul IN este folosit pentru a determina dacă o valoare face parte dintr-o listă de valori. Lista poate fi specificată ca valori literale, folosind o listă de valori separate prin virgule și încadrată între paranteze, sau poate fi selectată din baza de date folosind o *subselecție* (o *subinterogare*), care este o interogare în cadrul unei alte interogări.

Exemplu:

- Afișați toate filmele pentru care MOVIE_GENRE_CODE este Drama, Forgn sau Rmce. Evident, această interogare ar putea fi scrisă folosind trei condiții de egalitate, separate prin operatorul logic OR, dar este mult mai simplu să o scriem cu ajutorul operatorului IN:

```
SELECT MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE FROM MOVIE
WHERE MOVIE_GENRE_CODE IN ('Drama','Forgn','Rmce')
ORDER BY MOVIE_GENRE_CODE, MOVIE_TITLE;
```

EXISTS

Operatorul EXISTS este folosit pentru a determina dacă o subinterogare conține înregistrări. Dacă în setul de rezultate al subinterogării nu există, operatorul returnează valoarea logică „false”; dacă setul de rezultate conține cel puțin un rând, valoarea logică devine „adevărat”.

Exemplu:

Proprietarul magazinului a auzit că filmul *The Last Samurai* se închiriază bine atât în format DVD, cât și-n format VHS și vrea să se asigure că în inventarul magazinului există o copie VHS. Tabelul MOVIE_COPY conține un rând pentru fiecare copie a unui film din inventarul magazinului, așa că puteți folosi o subinterogare pentru a afla dacă există copii VHS ale filmului în inventar.

De cele mai multe ori, operatorul EXISTS este folosit în conjuncție cu o formă mai complexă de subinterogare, numită *subinterogare corelată* (*correlated subquery*), în care valorile datelor din interogarea externă (MOVIE_ID, în acest caz) sunt comparate cu rândurile din interogarea internă. Este suficient să știți că subinterogarea va returna un rând dacă există o copie VHS în stoc și nu va returna nici un rând dacă nu există o astfel de copie. Aceeași interogare ar putea fi scrisă și folosind operatorul IN sau folosind o *uniune*, care leagă rândurile din două tabele.

```
SELECT m.MOVIE_ID, m.MOVIE_TITLE FROM MOVIE m
WHERE m.MOVIE_TITLE = 'The Last Samurai' AND EXISTS
```

```
(SELECT c.MOVIE_ID FROM MOVIE_COPY c WHERE m.MOVIE_ID =  
c.MOVIE_ID)
```

Operatori aritmetici

În SQL, operatorii aritmetici sunt folosiți pentru efectuarea calculului matematic - la fel ca și-n formulele dintr-o foaie de calcul tabelar sau într-un limbaj de programare, precum Java sau C. Cei patru operatori aritmetici din SQL sunt:

Operator	Descriere
+	adunare
-	scădere
*	înmulțire
/	împărțire

Funcții SQL elementare

O *funcție* este un tip special de program, care returnează o singură valoare de fiecare dată când este apelată. Termenul provine de la conceptul matematic al unei funcții. În SQL, funcțiile necesită întotdeauna specificarea unei expresii, care deseori include numele unei coloane. Cel mai des, funcțiile sunt folosite în lista de coloane a unei instrucțiuni SELECT. Sunt apelate pentru fiecare rând prelucrat de interogare și, ca urmare, returnează o singură valoare pentru fiecare rând din setul de rezultate.

Funcții pentru caractere

Funcțiile pentru caractere sunt numite astfel deoarece manipulează date de tip text.

Concatenarea șirurilor de caractere

Funcția de *concatenare a șirurilor de caractere* reunește mai multe șiruri de caractere pentru a forma o singură valoare în rezultatele interogării. Funcția standard de concatenare a șirurilor de caractere din SQL este apelată cu două bare verticale (||), dar există și excepții, cum ar fi Microsoft SQL Server, care folosește semnul plus (+) pentru concatenarea șirurilor de caractere.

UPPER

Funcția UPPER transformă literele dintr-un șir de caractere în litere mari. Numerele și caracterele speciale sunt lăsate ca atare.

Exemplu:

- Afișați comediiile (MOVIE_GENRE_CODE = 'Comdy') scriind titlurile cu majuscule. Remarcați că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.

```
SELECT UPPER(MOVIE_TITLE) AS MOVIE_TITLE FROM MOVIE  
WHERE MOVIE_GENRE_CODE = 'Comdy';
```

Observație: La folosirea funcțiilor SQL în condițiile WHERE, în cele mai multe situații, pentru o coloană căreia îi este aplicată o funcție nu poate fi folosită indexarea. Ca urmare, în cazul tabelelor mari, utilizarea funcțiilor în condițiile WHERE poate duce la probleme de performanță cu adevărat memorabile. În concluzie, trebuie să specificați întotdeauna clauza ORDER BY dacă ordinea în care apar rezultatele este importantă.

LOWER

Funcția LOWER este inversa funcției UPPER - transformă literele dintr-un șir de caractere în litere mici.

Exemplu de utilizare a funcției LOWER:

- Afișați comediiile (MOVIE_GENRE_CODE = 'Comdy') scriind titlurile cu minuscule. Remarcați că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.

```
SELECT LOWER(MOVIE_TITLE) AS MOVIE_TITLE FROM MOVIE  
WHERE MOVIE_GENRE_CODE = 'Comdy';
```

SUBSTR

Funcția SUBSTR apare în majoritatea implementărilor SQL, dar uneori are un nume puțin

diferit. De exemplu, funcția se numește SUBSTRING în Microsoft SQL Server, Sybase Adaptive Server și MySQL, dar SUBSTR în Oracle și DB2, Funcția returnează o porțiune a șirului de caractere, în funcție de parametrii furnizați, care specifică numele coloanei, poziția de început a subșirului în datele coloanei și lungimea subșirului returnat (numărul de caractere). Deși este o utilizare mai puțin obișnuită, funcția SUBSTR acceptă și un șir de caractere literal în locul numelui unei coloane. Iată forma generală a funcției, urmată de un exemplu:

SUBSTR (*numele_coloanei, poziția_de_început, lungimea_subșirului*)

- în tabelul PERSON, unele persoane au al doilea nume în întregime, alții au numai inițiala. Afișați numele complet al persoanelor al căror nume de familie începe cu litera „B”, sub forma unui singur șir de caractere care conține prenumele, inițiala și numele de familie.

```
SELECT PERSON_GIVEN_NAME as SUBSTRING(PERSON_MIDDLE_NAME,1,1)
AS INITIALA,' ', PERSON_FAMILY_NAME AS NUME
FROM PERSON WHERE SUBSTRING(PERSON_FAMILY_NAME,1,1)='B'
```

LENGTH

Funcția LENGTH returnează lungimea unui șir de caractere. Rețineți că implementările Microsoft SQL Server și Sybase Adaptive Server folosesc numele LEN pentru versiunea proprie a acestei funcții.

Exemple:

- Afișați lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiți o bază de date Oracle, DB2 sau MySQL.
SELECT MOVIE_TITLE, LENGTH(MOVIE_TITLE) AS LENGTH FROM MOVIE WHERE MOVIE_ID = 1;
- Afișați lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiți o baza de date Microsoft SQL Server.
SELECT MOVIE_TITLE, LEN(MOVIE_TITLE) AS LENGTH FROM MOVIE WHERE LEN(MOVIE_TITLE)<10;

Funcții matematice

Funcțiile matematice manipulează valori numerice, în conformitate cu regulile matematicii.

Funcția **ROUND** este prezentată în detaliu, inclusiv cu un exemplu, apoi urmează un tabel cu funcțiile matematice existente în majoritatea implementărilor SQL.

ROUND

Funcția ROUND rotunjește o valoare la un număr specificat de zecimale. Valoarea numerică este furnizată prin primul parametru, iar numărul de zecimale prin cei de-al doilea. În continuare este prezentat formatul general al funcției ROUND, urmat de un exemplu.

ROUND (*expresie_numerică, nr_de_poziții_zecimale*)

- Care este costul mediu al unei copii a filmului *The Last Samurai*, rotunjit la două zecimale?
SELECT ROUND((RETAIL_PRICE_VHS + RETAIL_PRICE_DVD) / 2, 2) AS AVG_COST FROM MOVIE WHERE MOVIE_TITLE = 'The Last Samurai';

Tabelul care urmează prezintă funcțiile matematice cel mai des întâlnite. Pentru toate, sintaxa generală este aceeași:

NUME_FUNCTIE (*expresie*)

Funcție	Descriere
ABS	Valoarea absolută a unui număr dat
COS	Cosinusul trigonometric al unui unghi specificat în radiani
EXP	Valoarea exponențială a unui număr dat
POWE	Ridică un număr la o putere (numărul și puterea sunt furnizate ca parametri)
SIN	Sinusul trigonometric al unui unghi specificat în radiani
TAN	Tangenta trigonometrică a unui unghi specificat în <i>radiani</i>

Funcții de conversie

Funcțiile de conversie transformă date dintr-un tip de date în altul.

CAST

Funcția CAST transformă date dintr-un tip de date în altul. Rețineți că această funcție nu este implementată de DB2.

CAST (*expresie AS tip de date*)

- Afișați prețul pentru formatul DVD al filmului *The Last Samurai*, cu un simbol dolar în fața sumei. Valoarea numerică trebuie să fie convertită într-un șir de caractere pentru a putea fi concatenată cu o valoare literală conținând simbolul dolar.

```
SELECT CONCAT('$: ', CAST(RETAIL_PRICE_DVD AS char(6))) as Price
FROM MOVIE WHERE MOVIE_TITLE = 'The Last Samurai'
```

sau

```
SELECT NOW();
```

```
SELECT concat('Data: ', CAST(NOW() AS DATE));
```

Funcții de agregare și gruparea rândurilor

O funcție de agregare este o funcție care combină mai multe rânduri de date într-un singur rând. Tabelul următor prezintă funcțiile de agregare acceptate în majoritatea implementărilor SQL:

Funcție	Descriere
AVG	Calculează valoarea medie pentru o coloană sau o expresie.
COUNT	Numără valorile dintr-o coloană. Puteți folosi cuvântul cheie distinct pentru a număra valorile unice, în locul tuturor valorilor (rândurilor) dintr-o coloană.
MAX	Găsește valoarea maximă dintr-o coloană.
MIN	Găsește valoarea minimă dintr-o coloană.
SUM	Însumează valorile dintr-o coloană.

Exemple:

- Care este prețul mediu al unui DVD? Observați că funcțiile ROUND și AVG sunt imbricate, astfel încât să obțineți rezultatul în dolari și cenți.

```
SELECT ROUND(AVG(RETAIL_PRICE_DVD),2) AS AVG_PRICE FROM MOVIE;
```
- Câte filme există în tabelul MOVIE?

```
SELECT COUNT(*) AS NUM_MOVIES FROM MOVIE;
```
- Câte genuri diferite de filme sunt reprezentate în tabelul MOVIE? Observați folosirea cuvântului cheie DISTINCT, astfel încât sistemul DBMS să numere numai valorile MOVIE_GENRE_CODE unice.

```
SELECT COUNT(DISTINCT(MOVIE_GENRE_CODE)) AS NUM_GENRES FROM MOVIE;
```

Clauza GROUP BY

Așa cum ați observat, dacă folosiți o funcție de agregare, ca atare, într-o interogare, obțineți ca rezultat un singur rând din întreaga interogare. Este logic, deoarece sistemul RDBMS nu știe ce alte rezultate ați dori să obțineți, decât dacă-i spuneți acest lucru - și tocmai acesta este scopul clauzei GROUP BY. Aceasta cere sistemului DBMS să grupeze rândurile selectate de interogare pe baza valorilor din una sau mai multe coloane și să aplice funcția (sau funcțiile) de agregare fiecărui grup, returnând un rând pentru fiecare grup din setul de rezultate. Este ca și cum ați cere subtotaluri pentru fiecare departament, în locul unui singur total pentru întreaga companie, dar, așa cum ați văzut, funcțiile de agregare pot face multe alte lucruri, nu numai însumarea unor valori. Sistemul DBMS va ordona rândurile selectate de interogare după coloanele din clauza GROUP BY, așa că grupurile vor fi returnate în ordine ascendentă, exceptând cazul în care adăugați o clauză ORDER BY care specifică un alt mod de ordonare.

Exemplu:

- Afișați fiecare cod de gen, împreună cu numărul de filme asociate fiecărui cod.

```
SELECT MOVIE_GENRE_CODE AS GENRE, COUNT(*) AS COUNT FROM MOVIE
GROUP BY MOVIE_GENRE_CODE;
```

Operatori pentru interogări compuse

Uneori este util să rulăm interogări multiple și să combinăm rezultatele într-un singur set de rezultate.

UNION

Operatorul UNION adaugă rândurile din setul de înregistrări al unei interogări la cel al unei alte înregistrări și, în același timp, elimină rândurile duplicate, într-un mod similar cu cel al cuvântului cheie DISTINCT. Operația este permisă numai dacă interogările sunt compatibile din punctul de vedere al uniunii, ceea ce înseamnă că au același număr de coloane și că tipurile de date ale coloanelor corespondente sunt compatibile.

Exemplu:

- Afișați pe o singură coloană toate valorile nenule pentru taxa de închiriere și taxa de întârziere din tabelul MOVIE_RENTAL

```
SELECT RENTAL_FEE AS FEE FROM MOVIE_RENTAL
WHERE RENTAL_FEE IS NOT NULL UNION
SELECT LATE_OR_LOSS_FEE AS FEE1 FROM MOVIE_RENTAL
WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

UNION ALL

UNION ALL funcționează la fel ca și operatorul UNION, exceptând faptul că rândurile duplicate *nu sunt* eliminate.

INTERSECT

Operatorul INTERSECT găsește valorile selectate dintr-o interogare, care apar și într-o altă interogare. În esență, găsește intersecția valorilor din cele două interogări. Totuși, doar un număr mic de sisteme DBMS (cele mai importante fiind Oracle și DB2) implementează acest operator. Nu este implementat în Microsoft SQL Server sau MySQL.

EXCEPT

EXCEPT este operatorul standard ANSI/ISO care găsește diferențele dintre două seturi de rezultate, returnând, în esență, valorile din prima interogare care nu apar în cea de-a doua interogare. Foarte puține sisteme DBMS implementează acest operator. În unele implementări, precum Oracle, operatorul se numește MINUS, nu EXCEPT.

1.3.3. Interogări din tabele multiple

Uniuni (join)

O *uniune (join)* este o operație într-o bază de date relațională care combină coloane din două sau mai multe tabele în rezultatele unei singure interogări. O uniune apare de fiecare dată când clauza FROM a unei instrucțiuni SELECT specifică numele mai multor tabele.

Uniuni de egalitate (equijoin)

Avem o *uniune de egalitate (equijoin)*, numită și *uniune internă (innerjoin)*, atunci când legăm una sau mai multe coloane dintr-un tabel (de obicei, o cheie externă) cu coloane similare dintr-un alt tabel (de obicei, cheia primară), folosind condiția de *egalitate* (cu alte cuvinte, considerăm că acele coloane se potrivesc dacă valorile datelor sunt egale).

Aceasta este cea mai des folosită formă de uniune. Totuși, termenul *uniune de egalitate (equijoin)* este rareori folosit în afara mediilor academice, fiind preferate denumirile *uniune internă (innerjoin)* sau *uniune standard (standard join)*.

Există două modalități de specificare a coloanelor corespondente: folosind clauza WHERE sau folosind clauza JOIN. Clauza JOIN a fost adăugată relativ recent în standardul SQL, așa că programatorii mai vechi sunt obișnuiți cu metoda bazată pe clauza WHERE.

Realizarea uniunilor folosind clauza WHERE

Folosirea clauzei WHERE pentru unirea tabelelor seamănă cu folosirea acesteia pentru eliminarea rândurilor de care nu aveți nevoie din setul de rezultate. Totuși, există unele diferențe. În primul rând, în condiția WHERE comparați o coloană cu o altă coloană, nu o coloană cu o constantă sau o expresie. În al doilea rând, atunci când coloanele din cele două tabele au același nume (o soluție recomandată) trebuie să specificați numele complet al coloanelor (adică numele cu calificator), astfel încât motorul SQL să știe care dintre cele două coloane este referită. Cea mai simplă formă de calificator este chiar numele tabelului, separat cu un caracter punct de numele coloanei.

În continuare este prezentat un *exemplu* de uniune realizată prin clauza WHERE, cu numele coloanelor specificate complet, folosind numele tabelelor. Observați că interogarea selectează coloanele MOVIE_ID și MOVIE_TITLE din tabelul MOVIE și genul corespunzător (MOVIE_GENRE_DESCRIPTION) din tabelul MOVIE_GENRE.

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE, MOVIE_GENRE
WHERE MOVIE.MOVIE_GENRE_CODE=MOVIE_GENRE.MOVIE_GENRE_CODE
ORDER BY MOVIE_ID;
```

Observație: Folosirea numelor complete ale tabelelor pentru specificarea coloanelor poate fi obositoare și consumatoare de timp, mai ales deoarece numele tabelelor pot avea 30 sau mai multe caractere în sistemele DBMS moderne. Din aceasta cauză, în SQL este permisă folosirea pseudonimelor (*aliases*) pentru numele tabelelor.

Exemplul următor prezintă instrucțiunea anterioară, după adăugarea pseudonimelor pentru numele tabelelor. Deși nu era necesar, pseudonimele au fost adăugate și în lista de coloane din clauzele SELECT și ORDER BY, pentru a vă arăta cum sunt folosite.

```
SELECT A.MOVIE_ID, B.MOVIE_GENRE_DESCRIPTION AS GENRE, A.MOVIE_TITLE
FROM MOVIE A, MOVIE_GENRE B
WHERE A.MOVIE_GENRE_CODE=B.MOVIE_GENRE_CODE
ORDER BY A.MOVIE_ID;
```

Realizarea uniunilor folosind clauza JOIN

Clauza JOIN este scrisă ca o referință de tabel în clauza FROM și, în esență, combină lista de tabele din clauza FROM și condiția de legătură scrisă anterior în clauza WHERE într-o singură clauză.

Sintaxa generală a clauzei JOIN pentru o uniune internă:

```
nume_tabel [INNER] JOIN nume_tabel { ON condiție | USING (nume_coloană
[,nume_coloană...]) }
```

Observați cele două opțiuni. Clauza ON permite specificarea unei condiții similare cu cea din clauza WHERE folosită în exemplul anterior. Clauza USING, pe de altă parte, specifică numele coloanelor folosite pentru legarea rândurilor. Totuși, clauza USING funcționează numai atunci când coloanele pe care se face legătura au nume identice în ambele tabele. Iată câteva *exemple*:

- **JOIN cu condiție ON:**

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE JOIN MOVIE_GENRE ON
MOVIE.MOVIE_GENRE_CODE = MOVIE_GENRE.MOVIE_GENRE_CODE
ORDER BY MOVIE_ID;
```

- **JOIN folosind cuvântul cheie USING** (în locul condiției ON) - o scurtătură elegantă atunci când coloanele din cele două tabele au același nume.

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE JOIN MOVIE_GENRE USING (MOVIE_GENRE_CODE)
ORDER BY MOVIE_ID;
```

- **JOIN cu cheie externă pe mai multe coloane.** Aceasta interogare afișează lista cu copiile filmelor din tabelul MOVIE_COPY care nu au fost vândute (coloana DATE_SOLD conține o valoare nulă) și care au fost închiriate (uniune cu tabelul MOVIE_RENTAL) dar nu au fost

încă returnate (coloana RETURNED_DATE conține o valoare nulă).

```
SELECT MOVIE_ID, COPY_NUMBER, DUE_DATE
FROM MOVIE_COPY JOIN MOVIE_RENTAL USING (MOVIE_ID, COPY_NUMBER)
WHERE DATE_SOLD IS NULL AND RETURNED_DATE IS NULL;
```

Uniuni naturale

O *uniune naturală* (*natural join*) se bazează pe toate coloanele cu același nume din două tabele. În esență, toate uniunile de egalitate pe care le-ați văzut deja sunt uniuni naturale. Totuși, sintaxa unei uniuni naturale este mult mai simplă, deoarece nu este necesar să specificați o condiție sau o listă de coloane - se înțelege de la sine care sunt coloanele folosite.

Exemplu: uniunea tabelelor MOVIE și MOVIE_GENRE, rescrisă sub forma unei uniuni naturale:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE NATURAL JOIN MOVIE_GENRE
ORDER BY MOVIE_ID;
```

Uniuni externe (outer joins)

O *uniune externă* (*outer join*) - pentru care un nume mai potrivit ar fi uniune *inclusivă* - include în setul de rezultate și rândurile pentru care nu există legături din cel puțin unul dintre tabele. Atunci când există rânduri fără legături, datele selectate din tabelul în care nu a fost găsită o legătură primesc valoarea nulă. Există trei tipuri de bază:

- **Uniune externă către stânga (left outer join).** Returnează toate rândurile din tabelul din stânga (cel specificat primul în clauza JOIN), împreună cu toate rândurile din tabelul din dreapta pentru care poate fi găsită o legătură.
- **Uniune externă către dreapta (right outer join).** Returnează toate rândurile din tabelul din dreapta (cel specificat al doilea în clauza JOIN), împreună cu toate rândurile din tabelul din stânga pentru care poate fi găsită o legătură. În esență, o uniune externă către stânga poate fi rescrisă ca o uniune externă către dreapta inversând ordinea de specificare a tabelelor și înlocuind cuvântul cheie LEFT cu RIGHT.
- **Uniune externă completă (full outer join).** Returnează toate rândurile din ambele tabele. Acest tip de uniune este cel mai puțin probabil să fie acceptat de implementarea SQL pe care o aveți, deoarece sintaxa ei standard este mai nouă decât a celorlalte două. Este esențial să înțelegeți că această uniune nu este același lucru cu un produs cartezian, care leagă *fiecare* rând dintr-un tabel cu *fiecare* rând din celălalt tabel. O uniune externă completă (full outer join) leagă fiecare rând dintr-un tabel cu zero sau mai multe rânduri *corespondente* din celălalt tabel. În realitate, nu veți întâlni prea multe situații în care să folosiți o uniune externă completă, dar aceasta poate fi utilă dacă între două tabele aveți o relație opțională în ambele direcții.

Sintaxa generală pentru o uniune externă este:

```
nume_tabel {RIGHT | LEFT | FULL} [OUTER] JOIN nume_tabel
{ON condiție | USING (nume_coloana [, nume_coloana ...] ) }
```

Exemplu de uniuni externe:

- Afișați toate descrierile genurilor de filme, împreună cu filmele asociate fiecărui gen. Observați în setul de rezultate rândurile care nu au nici o valoare în coloana MOVIE_TITLE - acestea sunt genurile de filme care nu au filme asociate. Dacă implementarea DBMS nu acceptă uniuni realizate cu cuvântul cheie USING, modificați instrucțiunea astfel încât să utilizeze cuvântul cheie ON.

```
SELECT MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE FROM
MOVIE_GENRE LEFT OUTER JOIN MOVIE USING (MOVIE_GENRE_CODE);
```

Auto-uniuni (self joins)

O *auto-uniune* (*self join*) este o uniune a unui tabel cu el însuși. Acest tip de uniune poate părea ciudat la prima vedere, dar uneori există relații în care cheia primară și cheia externă se

află în același tabel. Acestea se numesc *relații recursive* și există o asemenea relație și în baza de date a magazinului de produse video.

Exemplu:

Tabelul EMPLOYEE are o coloană numită SUPERVISOR_PERSON_ID, care este o cheie externă către coloana PERSON_ID din același tabel. Este folosită pentru a lega fiecare angajat de șeful direct, care, desigur, este un alt angajat, ceea ce înseamnă că și șeful respectiv are o coloană în tabelul EMPLOYEE. Interogarea următoare afișează trei coloane din tabelul EMPLOYEE, inclusiv PERSON_ID și SUPERVISOR_PERSON_ID:

```
SELECT PERSON_ID, EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,  
SUPERVISOR_PERSON_ID FROM EMPLOYEE;
```

Acum să presupunem că magazinul trebuie să producă un raport cu diferențele de salariu dintre șefi și subordonați. Puteți lega înregistrarea fiecărui angajat de cea a șefului direct, astfel încât să obțineți plata pe oră a șefului. Iată interogarea care face acest lucru:

```
SELECT A.PERSON_ID, A.EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,  
B.EMPLOYEE_HOURLY_RATE AS SUPV_HOURLY_RATE  
FROM EMPLOYEE A JOIN EMPLOYEE B  
ON A.SUPERVISOR_PERSON_ID = B.PERSON_ID;
```

Iată și interogarea finală, care include calcularea diferenței de salariu și realizează o uniune cu tabelul PERSON pentru a obține numele angajaților:

```
SELECT A.PERSON_ID, C.PERSON_GIVEN_NAME AS FIRST_NAME,  
C.PERSON_FAMILY_NAME AS LAST_NAME,  
B.EMPLOYEE_HOURLY_RATE - A.EMPLOYEE_HOURLY_RATE AS  
RATE_DIFF  
FROM EMPLOYEE A JOIN EMPLOYEE B  
ON A.SUPERVISOR_PERSON_ID = B.PERSON_ID  
JOIN PERSON C ON A.PERSON_ID = C.PERSON_ID;
```

Uniuni încrucișate (cross joins)

O uniune încrucișată (*cross join*) nu este altceva decât sintaxa standard pentru un produs cartezian. Interogarea care unea tabelele MOVIE și MOVIE_GENRE și obținea un produs cartezian poate fi rescrisă ca mai jos, folosind o uniune încrucișată:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE  
FROM MOVIE CROSS JOIN MOVIE_GENRE ORDER BY MOVIE_ID;
```

Subinterogări

O caracteristică foarte puternică a limbajului SQL sunt *subinterogările* (numite și *subselecții*), care, așa cum sugerează și numele, se referă la o instrucțiune SELECT care conține o instrucțiune SELECT subordonată. De obicei, subinterogările sunt folosite în clauza WHERE, ca modalitate de limitare a rândurilor returnate în setul de rezultate al interogării externe. Aceasta poate fi o modalitate foarte flexibilă de selectare a datelor.

O regulă esențială de sintaxă este ca subinterogarea să fie încadrată de paranteze. O altă idee importantă pe care trebuie să o înțelegeți este că orice operație efectuată cu o subinterogare poate fi făcută și printr-o uniune.

Subinterogări necorelate

O *subinterogare necorelată* este o subinterogare în care interogarea internă nu face nici o referire la interogarea externă care o conține. Aceasta înseamnă că poate fi rulată mai întâi interogarea internă, apoi setul de rezultate obținut poate fi folosit în interogarea externă.

Exemplu:

- Afișați toate limbile în care nu există nici un film în inventarul magazinului de produse video.
SELECT LANGUAGE_CODE, LANGUAGE_NAME
FROM LANGUAGE WHERE LANGUAGE_CODE NOT IN
(SELECT DISTINCT LANGUAGE_CODE FROM MOVIE_LANGUAGE)
ORDER BY LANGUAGE_CODE;

Subinterogări corelate

O *subinterogare corelată* este o subinterogare în care interogarea internă referă valorile furnizate de interogarea externă. Aceste subinterogări sunt mult mai puțin eficiente decât subinterogările necorelate, deoarece interogarea internă trebuie să fie apelată pentru fiecare rând găsit de interogarea externă.

Exemplu:

- Proprietarul magazinului vrea să transmită prin poștă un cupon valoric tuturor clienților care au plătit mai mult de 15\$ pentru o singură tranzacție de închiriere.

```
SELECT DISTINCT CUSTOMER_ACCOUNT_ID
FROM CUSTOMER_TRANSACTION A
WHERE 15 < (SELECT SUM(RENTAL_FEE) FROM MOVIE_RENTAL B
WHERE A.TRANSACTION_ID = B.TRANSACTION_ID);
```

Vizualizări în linie

Foarte puține implementări, printre care Oracle, permit folosirea unei subinterogări în clauza FROM a unei interogări, într-o construcție numită *vizualizare în linie*, în esență, această construcție permite ca setul de rezultate al unei interogări să fie tratat ca și cum ar fi un tabel sau o vizualizare predefinită.

Exemplu:

- Această interogare află numărul maxim de închirieri ale unui singur film:
SELECT MAX(RENTAL_COUNT) AS MAX_RENTAL_COUNT FROM
(SELECT MOVIE_ID, COUNT(*) AS RENTAL_COUNT
FROM MOVIE_RENTAL GROUP BY MOVIE_ID);

1.3.4. Funcții SQL avansate

Funcții pentru caractere

Funcțiile pentru caractere operează asupra datelor de tip text.

REPLACE

Funcția REPLACE caută un șir de caractere și înlocuiește caracterele găsite cu caracterele din șirul de înlocuire. Iată sintaxa generală a funcției;

```
REPLACE (șir de caractere, șir căutat, șir de înlocuire)
```

- *șir de caractere* reprezintă șirul de caractere în care se face căutare și, în cele mai multe cazuri, este numele unei coloane dintr-un tabel, dar poate fi orice expresie care are ca rezultat un șir de caractere.
- *șir căutat* este un șir format dintr-un caracter sau mai multe, care trebuie căutate în *șir de caractere*.
- *șir de înlocuire* este șirul de caractere cu care se înlocuiește orice apariție a *șirului căutat* în *șir de caractere*.

Exemplu care înlocuiește toate liniuțele de despărțire, din numărul de telefon al unei persoane cu puncte:

```
SELECT PERSON_PHONE,
REPLACE(PERSON_PHONE, '-', '.') AS DISPLAY_PHONE FROM PERSON;
```

LTRIM

Funcția LTRIM elimină spațiile de la începutul unui șir de caractere. Rețineți că sunt eliminate *numai* spațiile de la începutul șirului - cele din mijlocul și de la sfârșitul șirului sunt lăsate ca atare.

Exemplu general:

```
LTRIM(' String with spaces ')
Returnează acest șir: 'String with spaces '
```

RTRIM

Funcția RTRIM este similară cu LTRIM, dar elimină spațiile de la sfârșitul unui șir de caractere.

Funcții pentru valori nule (NVL, ISNULL, IFNULL)

Oracle, Microsoft SQL Server și MySQL pun la dispoziție o funcție care înlocuiește valorile nule cu o valoare specificată. Din nefericire, fiecare implementare folosește propriul nume pentru această funcție: NVL în Oracle, ISNULL în SQL Server și IFNULL în MySQL. Se pare că în DB2 nu există o funcție echivalentă.

Exemplu:

```
SELECT IFNULL(LATE_OR_LOSS_FEE, 0) AS LATE_OR_LOSS_FEE FROM
MOVIE_RENTAL WHERE TRANSACTION_ID=9;
```

ASCII

Funcția ASCII returnează valoarea din setul de caractere ASCII (un număr între 0 și 255) pentru un șir format dintr-un singur caracter. De exemplu, codul ASCII pentru spațiu este 32, așa că ASCII(' ') returnează valoarea 32.

CHAR (CHR)

Funcția CHAR (numită CHR în Oracle și DB2) returnează caracterul corespunzător unei valori ASCII (un număr între 0 și 255). De exemplu, funcția ASCII(44) returnează o virgulă, deoarece codul ASCII pentru virgulă este 44. Această funcție este utilă în special pentru concatenarea caracterelor care nu pot fi afișate sau sunt greu de manipulat în SQL. Câteva dintre caracterele ASCII folosite frecvent cu această funcție sunt prezentate în tabelul următor. Puteți folosi funcția ASCII sau un tabel cu setul de caractere ASCII (ușor de găsit în Internet) dacă vreți să aflați și alte valori.

Valoare ASCII	Caracter
9	Tab
10	Linie nouă
13	Retur de car (CR)
39	Apostrof

Funcții matematice

Funcțiile matematice returnează rezultatul unei operații matematice și, de obicei, acceptă ca parametru de intrare o expresie numerică, care poate fi o valoare literală, o valoare numerică dintr-o coloană a unui tabel sau orice expresie (inclusiv rezultatul unei alte funcții) care produce o valoare numerică.

SIGN

Funcția SIGN primește ca argument o expresie numerică și returnează una dintre următoarele valori, în funcție de semnul numărului de intrare.

Valoare	Semnificație
-1	nr de intrare este negativ
0	nr de intrare este zero
1	nr de intrare este pozitiv
null	valoarea de intrare este nulă

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, SIGN(LATE_OR_LOSS_FEE) AS FEE_SIGN FROM
MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

SQRT

Funcția SQRT primește ca argument o expresie numerică și returnează rădăcina pătrată a acesteia. Sintaxa generală a funcției este:

SQRT (expresie_numerică)

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, SQRT(LATE_OR_LOSS_FEE) AS FEE_SQRT FROM
MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

CEILING (CEIL)

Funcția CEILING returnează cel mai mic întreg mai mare sau egal cu valoarea expresiei

numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjește numărul prin adăugire până la următorul număr întreg. Există câteva probleme interesante de compatibilitate legate de denumire între implementările SQL: Microsoft SQL Server folosește numele CEILING, Oracle folosește numele CEIL, în timp ce DB2 și MySQL permite folosirea ambelor nume (CEIL și CEILING).

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, CEILING(LATE_OR_LOSS_FEE) AS FEE_CEILING  
FROM MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

FLOOR

Funcția FLOOR este inversa logică a funcției CEILING - returnează cel mai mare întreg mai mic sau egal cu valoarea expresiei numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjește numărul prin scădere până la următorul număr întreg.

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, FLOOR(LATE_OR_LOSS_FEE) AS FEE_FLOOR  
FROM MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

1.3.5. Folosirea avantajelor oferite de vizualizări

O vizualizare (view) este o interogare stocată în baza de date, care pune la dispoziția utilizatorului bazei de date un subset particularizat de date din unul sau mai multe tabele ale bazei de date. Frumusețea vizualizărilor constă în faptul că după ce sunt create, pot fi interogate ca și cum ar fi tabele reale. De fapt, nu este nevoie ca utilizatorul să știe dacă folosește o vizualizare sau un tabel real. Mai mult, vizualizările oferă și alte avantaje:

- Maschează coloanele pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează rândurile pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează operațiile complexe, precum uniunile
- Cresc performanțele interogărilor (în unele sisteme RDBMS, cum ar fi Microsoft SQL Server). Sintaxa generală pentru crearea unei vizualizări este:

```
CREATE [OR REPLACE] VIEW nume_vizualizare AS interogare_sql;
```

1.4. ACTUALIZAREA DATELOR FOLOSIND LIMBAJUL SQL

Instrucțiunile DML (Data Manipulation Language), reprezintă o parte a limbajului SQL folosită pentru întreținerea datelor stocate în tabelele relaționale ale bazei de date.

Limbajul DML este format din trei comenzi SQL:

- **INSERT** Adaugă noi rânduri într-un tabel al bazei de date
- **UPDATE** Actualizează rândurile existente într-un tabel al bazei de date
- **DELETE** Șterge rânduri dintr-un tabel al bazei de date.

Observație: instrucțiunile DML individuale afectează datele dintr-un singur tabel. Atunci când o instrucțiune DML folosește o vizualizare, toate coloanele vizualizării referite în instrucțiunea DML trebuie să corespundă unor coloane dintr-un singur tabel fizic al bazei de date.

Instrucțiunile SQL din acest capitol presupun că sistemul DBMS pe care îl utilizați folosește formatul AAAA-LL-ZZ pentru datele calendaristice.

La formarea instrucțiunilor DML trebuie să țineți seama de următoarele aspecte referitoare la restricțiile tabelului modificat:

- **Restricții de tip cheie primară.** Atunci când inserați un nou rând într-un tabel, cheia primară a noului rând trebuie să fie unică în întregul tabel. Când modificați valoarea unei chei primare (cea ce se întâmplă rareori), noua valoare trebuie să fie unică în întregul tabel.
- **Restricții de unicitate.** Ca și în cazul cheilor primare, coloanele pe care a fost definită o

restricție de unicitate trebuie să aibă valori unice în întregul tabel.

- **Restricții referențiale.** Nu puteți insera sau actualiza valoarea unei chei externe decât dacă există deja rândul părinte corespondent care conține valoarea cheii în coloana cheii primare. În sens invers, nu puteți șterge un rând părinte dacă există rânduri subordonate care referă valoarea din rândul părinte, decât dacă restricția a fost definită cu opțiunea ON DELETE CASCADE. În general, inserările în tabele trebuie să fie făcute ierarhic (mai întâi rândurile părinte, apoi rândurile copii), iar ștergerile trebuie făcute în ordine inversă (copiii înaintea părinților).
- **Restricții NOT NULL.** În cazul instrucțiunilor INSERT, trebuie să specificați valori pentru toate coloanele cu restricții NOT NULL. În cazul instrucțiunilor UPDATE, nu puteți înlocui valorile unei coloane cu valori nule dacă pe coloana respectivă este definită o restricție NOT NULL. Dacă instrucțiunea DML referă o vizualizare, nu o puteți folosi într-o instrucțiune INSERT dacă una dintre coloanele obligatorii ale tabelului (o coloană cu o restricție NOT NULL) lipsește din definiția vizualizării.
- **Restricții de verificare (CHECK).** O instrucțiune INSERT sau UPDATE nu poate stoca într-o coloană o valoare care încalcă o restricție CHECK definită pentru coloana respectivă.

Instrucțiunea INSERT

Instrucțiunea SQL INSERT este folosită pentru inserarea noilor rânduri de date în tabele. Instrucțiunea are două forme de bază: una în care valorile coloanelor sunt specificate chiar în instrucțiune și alta în care valorile sunt selectate dintr-un tabel sau o vizualizare, folosind o subinterogare.

Inserarea unui singur rând de date folosind clauza VALUES

Instrucțiunea INSERT care folosește o clauză VALUES poate crea un singur rând la fiecare rulare, deoarece valorile pentru rândul de date respectiv sunt specificate chiar în instrucțiune.

Sintaxa generală a instrucțiunii este:

```
INSERT INTO nume_tabel sau vizualizare [ (listă_de_coloane) ] VALUES (listă_de_valori);
```

Reținem următoarele aspecte:

- Lista de coloane este opțională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, trebuie specificată o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca *întotdeauna* să includeți lista de coloane, deoarece omiterea acesteia face ca instrucțiunea INSERT să fie dependentă de definiția tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar și opțională, probabil instrucțiunea INSERT va eșua la următoarea rulare.
- Dacă lista de coloane este specificată, lista de valori trebuie să conțină o valoare pentru fiecare coloană din listă, în aceeași ordine. Cu alte cuvinte, între lista de coloane și lista de valori trebuie să existe o corespondență unu-la-unu. Orice coloană care lipsește din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în lista de valori pentru specificarea unei valori nule pentru o coloană.

Exemplu care conține două instrucțiuni INSERT, una care creează un nou rând în tabelul MOVIE, cu o listă opțională de coloane din care este omisă coloana RETAIL_PRICE_VHS, și alta care creează un rând în tabelul MOVIE_COPY pentru același timp, dar fără lista opțională de coloane:

```
INSERT INTO MOVIE
(MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE,
RETAIL_PRICE_DVD, YEAR_PRODUCED) VALUES (21, 'Drama', 'PG-13', 'Ray', 29.95, '2004');
INSERT INTO MOVIE_COPY VALUES (21, 1, '2005-04-01', null, 'V');
```

Inserări masive folosind o instrucțiune SELECT internă

O altă soluție, care poate fi folosită pentru a insera rânduri multiple într-un tabel, este forma care folosește o instrucțiune SELECT internă. Această formă este utilă și pentru stabilirea

următoarei valori disponibile pentru o cheie primară cu valori secvențiale, cum este coloana MOVIE_ID din tabelul MOVIE. De asemenea, poate fi folosită atunci când creați un tabel temporar pentru testare și vreți să-l populați cu toate datele dintr-un alt tabel.

Sintaxa generală a instrucțiunii este:

```
INSERT INTO nume_tabel sau vizualizare [(lista__ de__coloane)] SELECT  
instrucțiune_select;
```

Reținem următoarele aspecte:

- Lista de coloane este opțională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, instrucțiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca întotdeauna să includeți lista de coloane, deoarece omiterea acesteia face ca instrucțiunea INSERT să fie dependentă de definiția tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar și opțională, probabil instrucțiunea INSERT va eșua la următoarea rulare.
- Dacă lista de coloane este specificată, instrucțiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din listă, în aceeași ordine. Cu alte cuvinte, între lista de coloane și setul de rezultate al instrucțiunii SELECT trebuie să există o corespondență unu-la-unu. Orice coloană care lipsește din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în instrucțiunea SELECT pentru specificarea unei valori nule pentru o coloană.

Exemplu: să presupunem că toate filmele sunt acum disponibile și în limba franceză. Tabelul MOVIE_LANGUAGE conține rânduri pentru limba franceză pentru o parte din filme, așa că acum vrem să le adăugăm numai pe cele care lipsesc.

```
INSERT INTO MOVIE_LANGUAGE (MOVIE_ID, LANGUAGE_CODE)  
SELECT MOVIE_ID, 'fr' FROM MOVIE WHERE MOVIE_ID NOT IN  
(SELECT MOVIE_ID FROM MOVIE_LANGUAGE WHERE LANGUAGE_CODE = 'fr');
```

Exemplul următor inserează un nou rând în tabelul MOVIE, folosind instrucțiunea SELECT pentru a găsi valoarea maximă din coloana MOVIE_ID, pe care o incrementează cu o unitate, obținând astfel cheia primară a noului rând:

```
INSERT INTO MOVIE  
(MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE,  
RETAIL_PRICE_DVD, YEAR_PRODUCED) SELECT MAX(MOVIE_ID)+1, 'Drama', 'PG-13', 'The  
Terminal', 24.95, '2004' FROM MOVIE;
```

Instrucțiunea UPDATE

Instrucțiunea UPDATE este folosită pentru actualizarea datelor din coloanele unui tabel (sau ale unei vizualizări).

Sintaxa generală a instrucțiunii UPDATE este:

```
UPDATE nume_tabel sau vizualizare SET nume_coloană = expresie  
[, nume_coloană = expresie...] [WHERE condiție];
```

Reținem următoarele aspecte:

- Clauza SET conține o listă cu una sau mai multe coloane, împreună cu o expresie care specifică noua valoare pentru fiecare coloană. În esență, aceasta este o listă de perechi nume-valoare, separate prin virgule, cu un operator de egalitate între fiecare nume și valoare.
- *Expresia* poate fi o constantă, un alt nume de coloană sau orice altă expresie pe care SQL o poate transforma într-o singură valoare, care va fi apoi atribuită coloanei respective.
- Clauza WHERE conține o expresie care limitează rândurile actualizate. Dacă această clauză este omisă, motorul SQL va încerca să actualizeze toate rândurile din tabel sau din vizualizare.

Exemple:

- Un film adăugat cu MOVIE_ID = 21 a fost introdus cu un preț incorect. Prețul pentru copia VHS ar trebui să fie 29.95, iar prețul pentru copia DVD ar trebui să fie 34.95. Instrucțiunea următoare actualizează prețurile, introducând valorile corecte. Observați că instrucțiunea actualizează două coloane din același rând al tabelului MOVIE.

```
UPDATE MOVIE SET RETAIL_PRICE_VHS = 29.95, RETAIL_PRICE_DVD = 34.95
WHERE MOVIE_ID = 21;
```

- Proprietarul magazinului de produse video a decis să crească salariul tuturor funcționarilor (EMPLOYEE_JOB_CATEGORY = 'C') cu 8 procente. Această instrucțiune va actualiza o singură coloană de pe mai multe rânduri (toate rândurile pentru care corespunde categoria postului). Puteți folosi un calcul pentru a crește fiecare salariu cu 8 procente, înmulțind valoarea existentă cu 1.08 și rotunjind rezultatul la două cifre zecimale, cu ajutorul funcției ROUND.

```
UPDATE EMPLOYEE SET EMPLOYEE_HOURLY_RATE =
ROUND(EMPLOYEE_HOURLY_RATE * 1.08, 2) WHERE EMPLOYEE_JOB_CATEGORY = 'C';
```

Instrucțiunea DELETE

Instrucțiunea DELETE șterge unul sau mai multe rânduri dintr-un tabel. Instrucțiunea poate să folosească și o vizualizare, dar numai dacă aceasta se bazează pe un singur tabel (ceea ce înseamnă că instrucțiunile DELETE nu pot fi folosite pentru vizualizări care conțin uniuni). În instrucțiunile DELETE nu sunt referite niciodată coloane, deoarece instrucțiunea șterge rânduri întregi de date, inclusiv toate valorile datelor (toate coloanele) din rândurile afectate. Dacă vreți să ștergeți o singură valoare din rândurile existente, folosiți instrucțiunea UPDATE pentru a înlocui valorile respective cu valori nule (presupunând că valorile nule sunt permise în acele coloane). Sintaxa generală a instrucțiunii DELETE este: `DELETE FROM nume_tabel sau vizualizare [WHERE condiție];`

Reținem următoarele aspecte:

- Clauza WHERE este opțională. Totuși, este folosită aproape întotdeauna, deoarece o instrucțiune DELETE fără o clauză WHERE încearcă să șteargă *toate* rândurile din tabel - ceea ce, în majoritatea cazurilor, nu este rezultatul care se dorește.
- Atunci când este inclusă, clauza WHERE specifică rândurile care urmează să fie șterse. Orice rând pentru care condiția WHERE este evaluată ca adevărată este șters din tabel.
- Rețineți că nu puteți șterge rânduri dacă încălcați astfel o restricție referențială. În general, rândurile subordonate trebuie șterse înainte rândurilor părinte.

Exemple:

- Ștergeți filmul (cu MOVIE_ID = 21). Observați că trebuie să ștergeți mai întâi rândurile corespondente din tabelele MOVIE_COPY și MOVIE_LANGUAGE, deoarece acestea sunt rânduri „copii” ale rândului din tabelul MOVIE. Dacă încercați să ștergeți mai întâi rândul din tabelul MOVIE, sistemul DBMS va afișa un mesaj de eroare referitor la încălcarea unei restricții referențiale.

```
DELETE FROM MOVIE_COPY WHERE MOVIE_ID = 21;
DELETE FROM MOVIE_LANGUAGE WHERE MOVIE_ID = 21;
DELETE FROM MOVIE WHERE MOVIE_ID = 21;
```

- Ștergeți din tabelul MOVIE_LANGUAGE toate rândurile pentru limba spaniolă (LANGUAGE_CODE = 'es'). Rețineți că în multe implementări SQL se face diferențierea literelor mari de cele mici, caz în care valoarea codului de limbă trebuie specificată cu litere mici pentru a se potrivi cu datele din tabel.

```
DELETE FROM MOVIE_LANGUAGE WHERE LANGUAGE_CODE = 'es';
```

CAP.2. PROGRAMARE ORIENTATĂ OBIECT ȘI STRUCTURI DE DATE

2.1. CARACTERISTICILE GENERALE ALE PROGRAMĂRII ORIENTATE OBIECT

2.1.1. Principiile ce stau la baza programării orientate obiect

Programarea orientată obiect este unul din cei mai importanți pași făcuți în evoluția limbajelor de programare spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai natural ființei umane. Astfel unitățile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât modul de lucru al calculatorului.

Programarea obiect se desfășoară ca programarea modulară cu tipuri abstracte de date, în scopul creării posibilității de încapsulare a structurilor de date. Ea favorizează reutilizarea și extensibilitatea aplicațiilor.

Principalele concepte ce stau la baza programării orientate obiect sunt:

A) Abstractizarea.

Abstractizarea este una din căile fundamentale prin care noi, oamenii, ajungem să înțelegem și să cuprindem complexitatea, oferind posibilitatea ca un program să ignore unele aspecte ale informației pe care o manipulează, adică posibilitatea de a se concentra asupra esențialului. Fiecare obiect în sistem are rolul unui "actor" abstract, care poate executa acțiuni, își poate modifica și comunica starea și poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilități. Procesele, funcțiile sau metodele pot fi de asemenea abstracte, și atunci când sunt, sunt necesare o varietate de tehnici pentru a extinde abstractizarea.

O abstracțiune exprimă toate caracteristicile esențiale ale unui obiect, care fac ca acesta să se distingă de alte obiecte, oferind o definiție precisă a granițelor conceptuale ale obiectului.

✚ Comportament vs. Implementare.

În procesul de abstractizare atenția este îndreptată exclusiv spre aspectul exterior al obiectului, adică spre comportarea lui, ignorând implementarea acestei comportări. Cu alte cuvinte abstractizarea ne ajută să delimităm ferm "CE face obiectul" de "CUM face obiectul ceea ce face".

✚ Modelul Client-Server. Responsabilități. Protocol.

Comportarea unui obiect se caracterizează printr-o sumă de servicii sau resurse pe care el le pune la dispoziția altor obiecte. Un asemenea comportament, în care un obiect, numit server, oferă servicii altor obiecte, numite clienți, este descris de așa-numitul model client-server.

Totalitatea serviciilor oferite de un obiect server constituie un contract sau o responsabilitate a obiectului față de alte obiecte. Responsabilitățile sunt îndeplinite prin intermediul unor operații, fiecare operație a unui obiect caracterizându-se printr-o semnătură unică, formată din: nume, o listă de parametrii formali și un tip returnat. Mulțimea operațiilor unui obiect, împreună cu regulile lor de apelare constituie protocolul obiectului.

B) Încapsularea

Încapsularea este conceptul complementar abstractizării. Dacă rezultatul operației de abstractizare pentru un anumit obiect este identificarea protocolului său, atunci încapsularea are de a face cu selectarea unei implementări și tratarea acesteia ca pe un secret al respectivei abstracțiuni. **Încapsularea** – numită și *ascunderea de informații*, asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv); doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

Prin urmare, încapsularea este procesul în care are loc ascunderea implementării față de majoritatea obiectelor-client. Determinarea protocolului pentru un obiect trebuie să precedă decizia privind implementarea sa. Sintetizând putem defini încapsularea astfel: ***Încapsularea este procesul de compartimentare a elementelor care formează structura și comportamentul unei abstracțiuni; încapsularea servește la separarea interfeței "contractuale" de implementarea acesteia.***

Interfața vs. Implementare

Din definiția de mai sus rezultă că un obiect este format din două părți distincte: **interfața (protocolul)** și respectiv **implementarea acestei interfețe**. Abstractizarea este procesul prin care este definită interfața obiectului, în timp ce încapsularea definește reprezentarea (structura) obiectului împreună cu implementarea interfeței. Ascunderea structurii obiectului și a implementării metodelor sale este ceea ce se înțelege prin noțiunea de ascundere a informației.

Beneficiile încapsulării

- Separarea interfeței de reprezentarea unui obiect permite modificarea reprezentării fără a afecta în vreun fel pe nici unul din clienții săi, întrucât aceștia depind de interfață și nu de reprezentarea obiectului-server.
- Încapsularea permite modificarea programelor într-o manieră eficientă, cu un efort limitat și bine localizat.

Încapsularea aplică principiul abstracției: un obiect nu este accesibil pentru operațiile vizibile (interfața sa externă) și implementarea sa (structurile datelor asociate) este ascunsă. Astfel, modificările datelor rămân locale, la obiecte, fără să afecteze programele utilizatorilor.

Această încapsulare conferă independență între programe, operații și date. Un avantaj imediat este acela că diferitele programe de aplicații pot fi partajate pe aceleași obiecte fără a cunoaște mecanismul de intrare-ieșire (import/export). Principiul încapsulării poate fi uneori temporar lăsând un acces mai mult sau mai puțin liber datelor dintr-un obiect, făcând posibilă stabilitatea în timp a alegerii implementării datelor.

Accesul direct al datelor la obiecte este un plus de eficiență care apelează procedura impusă de încapsulare. Între timp când implementarea datelor este modificată (de exemplu se schimbă tipul lor), se va face modificarea în toate programele utilizatoare ale datelor, în mod direct prin programarea structurată. Decizia de încapsulare se va lua făcându-se un compromis între performanță și flexibilitate. Putem spune că cele două tipuri de orientări permit răspunsul parțial la problema proiectării structurate.

C) Modularitatea

Scopul descompunerii în module este reducerea costurilor prin posibilitatea de a proiecta și revizui modulele în mod independent. Clasele și obiectele obținute în urma abstractizării și încapsulării trebuie grupate și apoi stocate într-o formă fizică, denumită modul.

Modulele pot fi privite ca și containere fizice în care declarăm clasele și obiectele rezultate în urma proiectării la nivel logic. Modulele formează așadar arhitectura fizică a programului.

Modularizarea constă în divizarea programului într-un număr de module care pot fi compilate separat, dar care sunt conectate (cuplate) între ele. Limbajele care suportă conceptul de modul fac în același timp distincția între interfața modulului și implementarea sa.

Reguli Generale de Modularizare

1. Structura fiecărui modul trebuie să fie suficient de simplă pentru a putea fi complet înțeleasă.
2. Implementarea unui modul trebuie să depindă doar de interfețele altor module, respectiv trebuie să fie posibilă modificarea implementării unui modul fără a avea cunoștințe despre implementarea altor module și fără a afecta comportarea celorlalte module.

3. Detaliile sistemului care se presupune că se vor modifica independent vor fi plasate în module diferite.

4. Singurele legături între module vor fi acelea a căror modificare este improbabilă.

5. Orice structură de date este încapsulată într-un modul, ea putând fi accesată direct din interiorul modulului dar nu poate fi accesată din afara modului decât prin intermediul obiectelor și claselor conținute în acel modul.

Putem defini *modularitatea ca fiind proprietatea unui sistem care a fost descompus într-un set de module coezive și slab cuplate*.

D) Ierarhizarea

Abstractizarea este un lucru bun, dar în majoritatea aplicațiilor - excepție făcând doar aplicațiile banale - vom descoperi mai multe abstracțiuni decât putem cuprinde la un moment dat. Încapsularea ne ajută să tratăm această complexitate prin ascunderea interiorului abstracțiunilor noastre. Modularitatea ne ajută și ea, oferindu-ne o modalitate de a grupa abstracțiuni legate logic între ele. Toate acestea deși utile, nu sunt suficiente. Adesea un grup de abstracțiuni formează o ierarhie, iar prin identificarea acestor ierarhii, putem simplifica substanțial înțelegerea problemei. Ierarhizarea reprezintă o ordonare a abstracțiunilor.

✓ **Polimorfismul**

Este abilitatea de a procesa obiectele diferit în funcție de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate.

✓ **Moștenirea (ierarhia de clase)**

Moștenirea – organizează și facilitează polimorfismul și încapsularea permițând definirea și crearea unor clase specializate plecând de la clase (generale) care sunt deja definite - acestea pot împărtăși (și extinde) comportamentul lor fără a fi nevoie de redefinirea aceluiași comportament. Aceasta se face de obicei prin gruparea obiectelor în clase și prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moștenire permite construirea unor clase noi, care păstrează caracteristicile și comportarea, deci datele și funcțiile membru, de la una sau mai multe clase definite anterior, numite clase de bază, fiind posibilă redefinirea sau adăugarea unor date și funcții noi. O clasă moștenitoare a uneia sau mai multor clase de bază se numește clasă derivată.

Moștenirea definește o relație între clase în care o clasă împărtășește structura și comportarea definită în una sau mai multe clase (după caz vorbim de moștenire simplă sau multiplă). Moștenirea implică o ierarhie de tip generalizare/specializare, în care clasa derivată specializează structura și comportamentul mai general al clasei din care a fost derivată.

✓ **Agregarea (ierarhia de obiecte)**

Agregarea este relația între două obiecte în care unul dintre obiecte aparține celuilalt obiect. Agregarea redă apartenența unui obiect la un alt obiect.

2.1.2. Avantajele programării orientate obiect

Orientarea obiect aduce avantaje decisive cum ar fi: *modelarea obiectelor aplicațiilor, modularitatea, reutilizabilitatea și extensibilitatea codului care conduc la o mai mare productivitate, și dezvoltarea unei mari calități a aplicațiilor*.

Deoarece utilizatorii doresc ca programele lor să se comporte într-o manieră fiabilă și previzibilă, este important să se înțeleagă modul în care POO și condusă de evenimente va contribui la corecta structurare și modularizare a programului.

Într-o manieră globală, un program orientat obiect este un *ansamblu de obiecte care printr-un schimb de mesaje declanșează anumite operații sau metode, facilitându-le stările lor interne și returnându-le parametrii*.

Prin crearea în mod vizual a unei singure linii de cod, se obține un program funcțional care, deși banal, demonstrează toate elementele unei corecte proiectări orientate obiect și conduse de evenimente.

Obiectele permit reprezentarea în mod direct a entităților lumii reale și a relațiilor dintre entități. Se definesc concepte noi, ca tip, clasă, moștenire, obținându-se o însumare a avantajelor sistemelor de gestiune a bazelor de date, care interoghează eficient datele, și a limbajelor procedurale care permit calcule complexe.

În timp ce structura unei părți din variabile conține date și cealaltă funcții de tratare, se organizează programele în entități active compuse din structuri de date ce ascund modul de funcționare. Același nume de funcție poate fi utilizată pentru efectuarea de acțiuni similare la obiecte diferite, ceea ce permite constituirea unui limbaj abstract și prezentarea în maniere similare a obiectelor înainte diferențiate.

Avantajele utilizării POO se pot sintetiza astfel:

✚ **Ușurința proiectării și reutilizării codului:** odată ce este testată corectitudinea funcționării unor obiecte dintr-o aplicație, acestea vor putea fi folosite fără nici o problemă și în altă aplicație. Acest avantaj poate fi valorificat prin constituirea de biblioteci de obiecte. În ceea ce privește proiectarea, se facilitează descompunerea problemelor complexe în subprobleme simple, care pot fi ușor modelate cu ajutorul obiectelor.

Abstractizare - proiectanții pot obține o imagine de ansamblu, urmărind comportarea obiectelor și interacțiunile dintre ele, detaliile fiind îngropate în compoziția obiectelor. Programarea orientată-obiect constituie o bună metodă de organizare a programelor de calcul (software). Proprietățile POO conduc la un cod principal compact și elegant. Obiectele pot descrie mai bine conceptele pe care le reprezintă, fiind mai logice și intuitive decât modul tradițional, cu simple structuri de date.

✚ **Siguranța datelor** - abilitatea obiectelor de a se comporta ca niște “cutii negre”, de a putea fi folosite fără a se cunoaște compoziția lor, asigură confidențialitatea datelor utilizate și micșorează frecvența aparițiilor și efectul erorilor legate de manipularea greșită a tipurilor de date. Modelând realitatea complexă, tehnicile orientate obiect, pun accentul pe comportamentul datelor, încapsulând în conceptul de obiect atât datele, cât și operațiile posibile asupra lor.

✚ Din punct de vedere **educațional**, aplicarea POO conduce la formarea rapidă a unor concepte globale de funcționare a metodei elementului finit. POO permite construcția unor aplicații prin asamblarea unor module existente, la nivel global, reprezentând astfel o metodă modernă, logică și eficientă nu numai în dezvoltarea de programe, dar și în utilizarea și înțelegerea acestora.

✚ În programarea orientată obiect interfața este separată de implementare. Interfața este partea vizibilă a clasei, parte care trebuie înțeleasă de utilizatorul acesteia. Implementarea este partea ascunsă, internă clasei, care este importantă doar pentru autorul clasei. Pot exista una sau mai multe implementări pentru o aceeași interfață. O implementare satisface cerințele unei interfețe dacă comportamentul definit de interfață este realizat de implementare.

Pe lângă avantajul simplificării, separarea aduce un plus de flexibilitate pentru implementatori, deoarece mai multe implementări pot servi o aceeași interfață. Implementările pot să difere în ceea ce privește eficiența de timp, spațiu, prețul sau calitatea documentației puse la dispoziție, sau orice ale caracteristici non-funcționale. De asemenea o singură implementare poate să satisfacă mai multe interfețe. În acest caz implementarea conține o uniune de metode cerute de fiecare din interfețe.

Numeroase limbaje de programare includ tehnici POO. În lucrarea de față se folosește, pentru exemplificare, limbajul Visual C++, deoarece C++ reprezintă limbajul standard în lumea POO. C++ permite ca programarea procedurală și cea orientată-obiect să fie folosite împreună într-un limbaj destul de uniform.

Limbajul C++ prezintă și alte avantaje față de celelalte limbaje obiectuale, cum ar fi:

- ✓ C++ produce un cod cu un timp de execuție foarte eficient;
- ✓ existența multor compilatoare, inclusiv a celor gratuite;
- ✓ posibilitatea utilizării modulelor existente Fortran și C;
- ✓ legăturile cu sistemul de operare Unix (datorate limbajului C);

- √ largă utilizare și acceptare a limbajului C++ a produs numeroase implementări, pe platforme diferite, existând multe biblioteci și programe de calcul reutilizabile;
- √ posibilitatea de interfațare cu numeroase programe importante.

Caracteristici generale ale claselor în Visual C++

2.1.3.1. Conceptul de clasă și tipuri de clase

O clasă este schema, planul după care este creat un obiect. În interiorul unei clase sunt precizate *denumirile tuturor variabilelor și tipurile de valori pe care acestea le vor stoca* (numeric, șir de caractere, etc.), *alături de toate operațiile* (metodele) *pe care un obiect, construit din respectiva clasă, le va conține.*

Clasa reprezintă o *abstractizarea a elementelor (proprietăților și operațiilor) comune partajate de o mulțime de obiecte și descrie implementarea acestora.*

Folosind instrumentele moderne precum **Visual C++**, alcătuirea unui program este cu mult mai simplă. Mediul Visual C++ permite editarea, compilarea, executarea și depanarea unui program în interiorul aceluiași mediu.

Programarea orientată-obiect presupune *ordonarea obiectelor și a acțiunilor din lumea reală sub formă de clase care pot fi create, manevrate și distruse.*

Datele care alcătuiesc un obiect și funcțiile exercitate asupra acelui obiect sunt combinate pentru a forma o clasă, sau o descriere a acelui obiect.

Clasele pot moșteni funcționalitatea altor obiecte, putând fi cu ușurință adăugate clase noi care depășesc clasele existente.

O clasă trebuie să asigure o interfață, utilizată pentru manipularea obiectelor existente în acea clasă. Pe cât posibil, detaliile de implementare trebuie ascunse utilizatorului, ceea ce permite programatorului dotarea claselor complicate cu o interfață simplă, precum și modificarea detaliilor de implementare, dacă este necesar.

Deoarece structurile orientate obiect descriu integral clasa din care face parte obiectul respectiv, fiecare clasă poate fi reutilizată cu ușurință, întrucât datele și funcțiile descrise de clasa respectivă sunt integrate.

Deoarece implementarea unei clase poate fi ascunsă în spatele unei interfețe, detaliile de implementare ale respectivei clase sunt ușor de modificat fără a-i afecta pe utilizatorii acesteia, atâta timp cât interfața nu se modifică.

Atunci când se utilizează limbajul Visual C++, obiectele sunt descrise printr-o **clasă**. O clasă are două componente principale:

- **declarația clasei** - conține interfața clasei și informații privind datele membre ale respectivei clase. Interfața clasei este de obicei localizată într-un fișier antet cu extensia **.h**. Orice fișier din program care folosește clasa respectivă trebuie să folosească directiva **#include**, pentru ca declarația de clasă să fie adăugată fișierului sursă de către preprocesor;
- **implementarea clasei** - include toate funcțiile membru care au fost declarate ca parte a clasei. De obicei implementarea clasei este localizată într-un fișier având extensia **.CPP**.

Pentru a face clasele ușor de citit și de întreținut, se va descompune codul asociat în două tipuri de fișiere: *fișiere header* și *fișiere de cod*. Fișierele din prima categorie conțin *declarațiile atributelor și funcțiilor membru clasei* în cauză. Aceasta nu conține altceva decât structura clasei. Fișierele din a doua categorie conținând *codul sau implementarea* nu sunt altceva decât fișiere ASCII care conțin definiția fiecărei metode declarate în fișierele header corespunzătoare.

Fișierul header trebuie să conțină următoarele informații:

- numele fișierului trebuie să reprezinte clasa;
- descriere a ceea ce face clasa;
- structura clasei, cea care este implementată în porțiunea privată;
- funcțiile membru împărțite pe categorii;

- comentarii tactice și strategice ale funcțiilor membru și grupurile de funcții membru, chiar și de atribute.

Funcțiile membru trebuie să fie documentate în fișierele header prin așa numitele **comentarii tactice** (comentarii conținute într-o singură linie ce sunt utile noilor utilizatori ai clasei în scopul înțelegerii acesteia). **Comentariile strategice** sunt utilizate în documentarea grupurilor sau blocurilor de funcții aflate într-o anumite legătură unele cu altele.

Fiecare funcție membru declarată în fișierul header trebuie să-și găsească implementarea corespunzătoare în fișierul de cod asociat clasei. Mai mult, fișierul de cod este amplasat acolo unde detaliile de modificare ale programului pot fi documentate.

C++ permite definirea de funcții având același nume, dar tipuri și număr diferite de parametri. Tipul valorilor returnate de astfel de funcții este irelevant în identificarea variantei apelate. Aceasta este o formă ad-hoc de polimorfism, cunoscută sub denumirea de **redefinirea funcțiilor**.

O altă formă de polimorfism acceptată în C++ este **redefinirea operatorilor**. De fapt, aceasta nu este altceva decât o variantă restrânsă a primei forme de polimorfism ad-hoc. Prin această metodă, este posibilă redefinirea operatorilor în vederea operării acestora asupra tipurilor de date definite de utilizator, fiind o soluție deosebit de importantă în toate aplicațiile cu un bogat suport matematic, în special.

Redefinirea funcțiilor și operatorilor poate crește gradul de înțelegere a codului. Totuși, este important să nu se uite că utilizarea excesivă a acestei tehnici poate conduce la obținerea unui cod mult mai criptat decât este cazul. În cazul redefinirii funcțiilor, se vor utiliza nume care să aibă sens în cadrul problemei.

Privitor la diferitele clase care pot apărea într-un program, există două modalități de organizare a claselor:

- √ **ierarhii** - o ierarhie poate fi folosită pentru organizarea claselor după modelul unui arbore genealogic, cu clase care moștenesc datele și funcțiile de la predecesorii lor;
- √ **categoriile** - adesea, un set de clase se folosesc împreună și prezintă un fel de dependență sau o altă categorie de relații între ele. Aceste clase pot fi organizate într-o categorie, pentru a simplifica utilizarea lor.

În majoritatea limbajelor orientate obiect, clasele pot moșteni proprietățile unei **clase de bază**. O clasă de bază are anumite proprietăți generale care sunt incluse în clasele derivate din acestea.

Organizarea claselor într-o ierarhie înainte de începerea scrierii declarațiilor de clasă este o modalitate de a căuta punctele comune ale claselor. Prin organizarea claselor, se pot determina cu ușurință relațiile care există între toate clasele, ceea ce poate contribui la selectarea clasei de bază și la determinarea claselor care trebuie create.

În multe cazuri, între clasele care lucrează împreună, există o strânsă interdependență. Clasele între care există legături strânse sunt adesea grupate în categorii care sunt frecvent folosite în grup; în multe cazuri nu are nici un rost utilizarea unei clase care este componentă a unui grup mai mare.

C++ suportă atât moștenirea simplă cât și pe cea multiplă, însă dintre cele două, cea simplă este cea mai utilizată în cadrul proiectelor C++, aplicarea moștenirii multiple fiind mult mai dificilă. Utilizarea moștenirii în C++ poate ajuta la creșterea productivității, dacă este aplicată corespunzător.

Clasele identificate drept clase predecesor, sau clase de bază, conțin date și funcții membru care sunt incluse în toate clasele descendente, sau derivate.

Când se folosește conceptul de moștenire, o clasă de bază stabilește un fel de contract care trebuie respectat de către toate subclasele. Dacă o operație poate fi executată utilizând o clasă de bază, toate subclasele trebuie să fie capabile să execute acea operație. Cu alte cuvinte, *într-o structură cu adevărat orientată-obiect se impune posibilitatea înlocuirii unei subclase cu o clasă de bază*. Acesta este cunoscut sub numele de **principiul de înlocuire al lui Liskov**.

Unul din obiectivele moștenirii este acela de a partaja un cod între mai multe clase, ceea ce nu face altceva decât să crească factorul de reutilizare al codului. În general, este bine de evitat ierarhiile de moștenire stufoase.

Principiile de ascundere a informației și încapsulare sunt suportate de Visual C++. Acest limbaj implementează aceste două noțiuni prin intermediul cuvintelor cheie **public** și **private**. Astfel, membrii clasei care sunt declarați în *secțiunea privată* sunt vizibili numai instanțelor și membrilor clasei, fiind în același timp inaccesibili celorlalte clase și instanțe ale acestora.

Secțiunea privată este utilizată în primul rând pentru informațiile stării unui obiect. Ea conține declarații ale datelor ce reprezintă atributele unui obiect. Această secțiune poate conține, de asemenea, declarațiile funcțiilor membru care nu sunt oferite spre utilizare clienților clasei.

Membrii claselor care sunt declarați în *secțiunea publică* a acesteia sunt accesibili tuturor obiectelor și funcțiilor, secțiunea publică constituind implementarea serviciilor pe care o clasă le oferă clienților săi.

Pentru apelul funcțiilor membre publice sau pentru accesul la datele publice ale unui obiect, din funcții care nu sunt membre, se folosesc operatorii de selecție (.) și (->), ca în cazul structurilor și uniunilor din C.

Un caz special al paradigmei client/server ne este oferit de C++, respectiv cazul *relației dintre o clasă dată și derivatele sale*, unde este posibil să se definească zone protejate. Membrii acestor zone se comportă ca membrii publici în clasele derivate, astfel aceste elemente violează principiul de ascundere a informației.

Numele tuturor funcțiilor membru sunt prefixate de numele de clasă ale acestora, aceasta fiind unica modalitate prin care compilatorul poate determina clasa căreia îi aparține funcția respectivă. Unica excepție de la regulă o constituie o funcție definită în interiorul declarației de clasă.

Constructori și destructori

În cazul variabilelor obișnuite, compilatorul asigură alocarea spațiului de memorie și eventual inițializarea explicită cu valori inițiale în declarație. Pentru variabilele dinamice, compilatorul C nu dispune de nici o metodă de inițializare și nici operatorul **new** nu rezolvă toate situațiile. În acest caz, rămâne în grija programatorului atribuirea de valori adecvate datelor înainte de utilizare. Această abordare este nesatisfăcătoare în multe situații în cazul obiectelor.

Pentru crearea, inițializarea, copierea și respectiv distrugerea obiectelor, în C++ se folosesc funcții speciale, numite constructori și destructori.

Un **constructor**, numit uneori și ctor, este o funcție membru specială, creată la crearea unui obiect al unei clase. Constructorul se apelează automat la crearea fiecărui obiect al clasei, static, automatic sau dinamic (cu operatorul new), inclusiv pentru obiecte temporare. Un obiect se creează atunci când este definit, sau când este creat din punct de vedere dinamic.

Utilitatea constructorilor este evidentă cel puțin sub două aspecte:

- ◆ constructorul asigură inițializarea corectă a tuturor variabilelor membru ale unui obiect;
- ◆ constructorul oferă o garanție că inițializarea unui obiect se va realiza exact o dată.

De multe ori este util să existe mai multe moduri de inițializare a obiectelor unei clase. Acest lucru se poate realiza furnizând diferiți constructori. Atâta timp cât constructorii diferă suficient în tipurile argumentelor lor, compilatorul le poate selecta corect, unul pentru fiecare utilizare.

Un **destructor**, cunoscut și sub numele de dtor, este o funcție membru specială care este apelată la distrugerea unui obiect. Destructorul este declarat ca neavând tip de valoare returnată și nu este declarat niciodată cu o listă de parametrii. Numele destructorului este alcătuit din numele clasei, precedat de un caracter tilda (~). Nu este absolut necesară definirea unui destructor, decât dacă există operații specifice care trebuie efectuate pentru a face curățenie, cum ar fi eliberarea resurselor unui sistem care au fost alocate.

Destructorul este apelat automat la eliminarea unui obiect, la încheierea timpului sau de viață, sau poate fi solicitat prin program, cu operatorul delete.

Constructorii și destructorii se declară și se definesc similar cu celelalte funcții membre, dar se disting de acestea printr-o serie de caracteristici specifice:

- numele funcțiilor constructor sau destructor coincide cu numele clasei căreia îi aparțin;
- în declarație și definiție nu se specifică nici un tip de rezultat, nici măcar void;
- constructorii pot avea parametri, inclusiv parametri implicați, și pot fi supradefiniți; destructorii nu au aceste proprietăți;
- nu se pot utiliza pointeri către constructori sau destructori;
- dacă o clasă nu dispune de constructori și destructori definiți, compilatorul va genera automat un constructor implicit, respectiv un destructor, funcții publice.
- constructorii sunt lansați în ordinea declarării obiectelor iar destructorii în ordine inversă.

Altă categorie de constructor este constructorul de copiere. Necesitatea de copiere a obiectelor intervine când un obiect este transferat ca parametru sau rezultat al unei funcții, sau la crearea unui obiect temporar. Soluția oferită de C++ constă în utilizarea unui constructor special, numit constructor de copiere. În absența unei definiții explicite în cadrul clasei, compilatorul generează automat un constructor de copiere care inițializează datele noului obiect cu valorile corespunzătoare din obiectul specificat, prin copiere membru cu membru. Aceasta nu este o soluție bună în cazul în care clasa conține membrii variabile dinamice, caz în care este necesară scrierea unui constructor de copiere special. Declarația constructorului de copiere pentru o clasă trebuie să specifice un parametru unic de tipul referință de obiecte a acelei clase.

Supraîncărcarea funcțiilor membre și a operatorilor

Funcțiile membre ale unei clase pot fi supraîncărcate. Mecanismul nu diferă de cel al supraîncărcării funcțiilor independente.

Programele manipulează obiecte care sunt reprezentări concrete ale unor concepte abstracte. De exemplu, datele de tip "int" din C++, împreună cu operatorii +, -, *, /, etc., furnizează o implementare restrictivă a conceptului matematic de întregi. Astfel de concepte includ de obicei un set de operatori care reprezintă operațiile de bază asupra obiectelor.

Clasele furnizează o facilitate de reprezentare a obiectelor neprimitive împreună cu un set de operații care pot fi efectuate cu astfel de obiecte. Se permite programatorului să furnizeze o notație mai convențională și mai convenabilă pentru a manipula obiectele unei clase. Acest lucru se realizează prin supraîncărcarea operatorilor. Trebuie evidențiat faptul că operatorii sunt deja supraîncărcați, fie pentru a putea opera asupra mai multor tipuri de bază (de ex. operatorii aritmetici admit ca operand orice tip numeric), fie pentru a efectua mai multe operații matematice (de ex. * este asociat înmulțirii dar și referirii datelor de tip pointer). Operația adecvată este selectată de compilator în funcție de tipul operanzilor.

Procedeul de supraîncărcare constă în definirea unei funcții cu numele operator simbol unde:

- operator este cuvântul cheie dedicat
- simbol este simbolul oricărui operator C++, mai puțin următorii: . (punct) .* :: ?:
intaxa supraîncărcării unui operator :

TipNumeClasa::operatorNumeOperator (ListaArgumente)

```
{  
    BlocInstructiuni  
}
```

Funcții virtuale și clase de bază abstracte

Cuvântul cheie virtual este utilizat în header-ul unei clase de bază în vederea declarării funcțiilor care definesc comportamentul tuturor claselor derivate din această clasă de bază. O funcție membru virtuală trebuie să posedă un corp, acesta situându-se în fișierul de cod asociat clasei. Acest mecanism permite implementarea polimorfismului.

Când se folosește o funcție virtuală, compilatorul construiește un tabel special, denumit **tabel de funcții virtuale** (virtual function table) sau `vtabl`, care se folosește pentru a păstra evidența funcțiilor care trebuie apelate pentru fiecare obiect din clasa respectivă. Când este apelată o funcție virtuală, `vtab`-ul este folosit pentru accesarea indirectă a funcției corecte.

Suprasarcina (overhead) adăugată prin utilizarea tabelului de funcții virtuale este destul de redusă, dar poate deveni semnificativă dacă se dispune de mii de obiecte mici sau dacă viteza de execuție este hotărâtoare. Din acest motiv, o funcție trebuie precizată ca fiind virtuală, deoarece acest caracter nu este prestabilit.

Principalul, avantaj al utilizării funcțiilor virtuale poate fi observat în momentul rulării, sistemul putând determina tipul obiectului din care provine funcția membru apelată.

O funcție virtuală pură este o funcție virtuală fără corp. Aceste funcții sunt declarate în clasele de bază. Derivarea unei clase dintr-o clasă de bază care conține cel puțin o funcție virtuală pură presupune declararea și definirea acestor funcții în cadrul clasei derivate, în caz contrar producându-se o eroare de compilare.

Utilizarea acestor funcții autorizează presupunerea că o clasă de bază a fost declarată ca având un comportament normal dar trebuie implementat în toate cazurile speciale. Este nevoie de ceva mult mai puternic pentru a asigura implementarea funcțiilor de către clienți (în acest caz clase derivate).

Scrierea de funcții virtuale pure în clasa de bază este echivalentă cu adoptarea unei asigurări: clasa de bază știe că viitorii săi clienți trebuie să implementeze aceste funcții, iar clienții sunt obligați să o facă.

O clasă de bază abstractă (CAB) este o clasă care are cel puțin o funcție membru virtuală pură. Clasele de bază abstracte nu pot avea instanțe.

Întotdeauna se va încerca să se găsească cât mai multe clase abstracte atunci când se proiectează un produs software utilizând tehnicile orientate obiect. O clasă de bază abstractă apare dintr-un set de clase care partajează attribute și/sau comportament util. Dacă se descoperă un comportament care este partajat de un număr de clase, va trebui proiectată o superclasă pentru capturarea acestuia într-un singur loc.

Un alt avantaj al claselor abstracte de bază îl constituie faptul că pot ține locul oricăreia din clasele derivate. Cu alte cuvinte, având o colecție de clase abstracte de bază, este mult mai ușor să se clasifice clasele, deoarece majoritatea noilor clase pot fi considerate ca fiind derivate ale cel puțin unei clase abstracte.

Este posibilă derivarea de clase atât din cele abstracte cât și din cele concrete. Combinarea corectă a celor două tipuri de clase va permite obținerea **ierarhiilor de clase C++**. Regulile de bază sunt:

- ◆ alegerea unei clase abstracte de bază pentru a servi drept clasă de bază tuturor celorlalte clase (abstracte sau nu). Clasele abstracte de bază vor deriva direct din aceasta;
- ◆ este posibilă crearea unei clase de bază abstracte prin derivarea dintr-o altă clasă de bază abstractă;
- ◆ clasele concrete sunt proiectate pentru a fi instanțiate. Acestea pot proveni din derivare, fie dintr-o clasă abstractă fie dintr-una concretă;
- ◆ niciodată nu se va deriva o clasă abstractă dintr-una concretă.

Polimorfismul

Polimorfismul este cel mai adesea folosit în programarea orientată obiect pentru a desemna funcțiile sau operatorii care pot fi aplicați mai multor categorii diferite de obiecte. Un aspect al polimorfismului este capacitatea utilizării unei funcții în mai multe obiecte diferite, fără a fi necesar un nume nou de funcție pentru fiecare obiect.

Limbajul Visual C++ permite “supraîncărcarea” numelui unei funcții, ceea ce înseamnă că aceasta poate avea mai multe definiții. Compilatorul determină funcția care trebuie apelată în funcție de lista de parametri.

Domeniu

Domeniul unei variabile se referă la vizibilitatea acesteia într-un program. Un identificator folosit pentru a denumi o variabilă sau o funcție, are un anumit domeniu atunci când este creat, iar acest domeniu determină modul și locul în care poate fi folosit respectivul nume. Dacă o variabilă se află în domeniu într-un anumit punct din program, atunci este vizibilă și poate fi folosită în majoritatea cazurilor. Dacă variabila se află în afara domeniului atunci nu este vizibilă și programul nu o va putea folosi.

În general, un program trebuie să folosească variabile al căror domeniu să fie cât mai redus posibil. Cu cât domeniul este mai restrâns, cu atât sunt mai mici șansele unei utilizări accidentale a unui identificator sau expunerii acestuia la efecte colaterale. De exemplu, întotdeauna este mai indicat transferul obiectelor ca parametrii ai unei funcții decât utilizarea variabilelor globale. Utilizarea variabilelor locale pentru o anumită funcție contribuie la capacitatea de reutilizare a funcției și elimină problemele de sincronizare într-un mediu cu execuții simultane cum este Windows.

Domeniul de definiție al unui nume prin care se identifică o entitate a unui program este zona din program în care numele este cunoscut și poate fi folosit. Dat fiind că un nume este făcut cunoscut printr-o declarație, domeniile numelor se diferențiază în funcție de locul în care este introdusă declarația în program. Un nume declarat într-un bloc este *local* blocului și poate fi folosit numai în acel bloc, începând din locul declarației și până la sfârșitul blocului și în toate blocurile incluse după punctul de declarație.

Argumentele formale ale unei funcții sunt tratate ca și când ar fi declarate în blocul cel mai exterior al funcției respective. Un nume declarat în afara oricărui bloc are ca domeniu fișierul în care a fost declarat și poate fi utilizat din punctul declarației până la sfârșitul fișierului. Numele cu domeniu fișier se numesc nume *globale*.

Un nume este vizibil în întregul său domeniu de definiție dacă nu este redefinit într-un bloc inclus în domeniul respectiv. Dacă într-un bloc interior domeniului unui nume se redefinește (sau se redeclară) același nume, atunci numele inițial (din blocul exterior) este parțial ascuns, și anume în tot domeniul redeclarării. Un nume cu domeniu fișier poate fi accesat într-un domeniu în care este ascuns prin redefinire, dacă se folosește operatorul de rezoluție pentru nume globale (::). Redefinirea unui nume este admisă numai în domenii diferite. Dacă unul din domenii este inclus în celălalt domeniu, atunci redefinirea provoacă ascunderea numelui din domeniul exterior în domeniul interior. Redefinirea în domenii identice produce eroare de compilare.

O entitate este creată atunci când se întâlnește definiția sa (care este unică) și este distrus (în mod automat) când se părăsește domeniul ei de definiție. O entitate cu nume global se creează și se inițializează o singură dată și are durata de viață până la terminarea programului. Entitățile locale se creează de fiecare dată când execuția programului ajunge în punctul de definire a acestora, cu excepția variabilelor locale declarate de tip static, care se inițializează o singură dată la prima execuție a instrucțiunii. O variabilă globală sau statică neinițializată explicit, este inițializată automat cu 0.

Domeniul unui identificator intră în acțiune la fiecare declarație a unui identificator. Tipurile de domenii disponibile într-un program C++ sunt următoarele:

• Domeniu local

Există mai multe categorii de domenii locale, cel mai simplu exemplu de domeniu local reprezentându-l o variabilă sau un alt obiect care este declarat în exteriorul oricărei funcții. Când variabila se află în domeniu, de la punctul în care a fost declarată și până la finalul fișierului sursă, acest tip de domeniu local este denumit **domeniu fișier**. Toate declarațiile care survin în exteriorul definițiilor de clase sau funcții au acest tip de domeniu.

Variabilele declarate în interiorul corpului unei funcții au domeniu local. Acestea sunt vizibile numai în interiorul funcției.

O altă categorie de domenii locale este **domeniul bloc**, unde o variabilă din interiorul unei instrucțiuni compuse sau al unui alt bloc este vizibilă până la capătul blocului. Variabilele

declarate în interiorul instrucțiunilor condiționale au de asemenea domeniu tip bloc.

- **Domeniu tip prototip de funcție** - acest domeniu există numai în interiorul unui prototip de funcție, fiind extrem de limitat. O variabilă cu astfel de domeniu, este în realitate numai o variabilă definită într-un prototip de funcție, al cărei domeniu se încheie la finalul prototipului.
- **Domeniu tip funcție** - acest tip de domeniu este rar întâlnit. Conceptul care a stat la baza sa este desemnarea etichetelor declarate în interiorul definiției unei funcții. Limbajul C++ nu acceptă saltul la o etichetă situată în exteriorul funcției curente și respectiv că același nume de etichete pot fi folosite în mai multe funcții.
- **Domeniu tip clasă** - numele unui membru tip clasă, uniune sau structură este strâns asociat cu clasa respectivă și are domeniu tip clasă. Un identificator cu domeniu tip clasă poate fi folosit în interiorul clasei, uniunii sau structurii. Dacă numele unei clase sau al unei variabile este utilizat pentru a controla accesul la un identificator, atunci acesta este vizibil și în afara clasei.

✚ Durata de viață a unui identificator

Într-un program Visual C++, fiecare variabilă sau obiect are o anumită durată de viață, care este separată de vizibilitatea sa. Astfel, programatorul are posibilitatea controlării momentului creării și a distrugerii unei variabile.

Durata de viață a unui identificator este foarte importantă în conceperea unui program. Obiectele de dimensiuni mari pot fi costisitor de creat, dar și de distrus.

O variabilă poate fi ascunsă de către altă variabilă având același nume, dar nu un alt domeniu, ceea ce înseamnă că este posibil ca o variabilă să existe și în afara domeniului propriu. Aceasta se întâmplă deseori atunci când numele unei variabile este folosit în domenii tip bloc sau clasă și maschează o altă variabilă care folosește aceleași nume.

O variabilă declarată static într-o funcție este creată la începutul programului și nu este distrusă înainte de încheierea acestuia. Acest fapt este util când se dorește ca variabila sau obiectul să-și amintească valoarea între două apelări ale funcției.

2.1.3.2. Clasele de bază MFC

Biblioteca de clase MFC poate fi împărțită pe trei nivele. La primul nivel se află **clasele care descriu comportamentul obiectelor grafice și încapsulează funcțiile API** (Application Program Interface).

Aceste obiecte (cum ar fi de exemplu ferestrele, butoanele, fonturile, contextul dispozitiv, etc.) sunt create și accesate de către programatorii Windows prin intermediul unor funcții sistem. Prin urmare clasele din MFC de la acest nivel încapsulează practic obiectele și funcțiile care le sunt atașate existente în API Windows, oferind un grad ridicat de abstractizare.

La al doilea nivel se află **clasele care nu depind de sistemul de operare Windows și care implementează structuri fundamentale de date** (liste, tablouri, dicționare, șiruri de caractere, dată calendaristică etc.).

Un mediu de lucru al aplicațiilor este o colecție de componente soft (clase, macro-uri, funcții globale) care pot fi utilizate în realizarea unei aplicații generice.

Biblioteca de clase MFC include un mare număr de clase corespunzătoare programării în Windows. Majoritatea acestor clase sunt derivate din CObject, clasă care se află la rădăcina ierarhiei de clase din MFC. Suplimentar, orice clasă care reprezintă o fereastră sau un control este derivată din clasa CWnd, care manevrează funcțiile de bază comune tuturor ferestrelor.

Clasele CObject și CWnd utilizează funcțiile virtuale, care permit programului să acceseze funcții de uz general, prin intermediul unui pointer de bază. Aceasta permite utilizarea cu ușurință a oricărui obiect derivat din CObject sau CWnd la interacțiunea cu cadrul de lucru MFC.

□ Clasa de bază CObject

Aproape orice clasă utilizată într-un program MFC este derivată din CObject. Clasa

CObject asigură patru tipuri de servicii:

- **diagnosticarea gestiunii memoriei furnizează mesaje de diagnostic la detectarea unor pierderi de memorie** (memory leak). Aceste pierderi sunt adeseori provocate de eșuarea operației de eliberare a unor obiecte create dinamic;
- **suportul creării dinamice folosește clasa CRuntimeClass pentru a permite crearea obiectelor la rularea programului**. Această operație este diferită de crearea dinamică a obiectelor utilizând operatorul new;
- **suportul de serializare permite stocarea și încărcarea unui obiect într-o modalitate orientată-obiect**;
- **informația de clasă runtime** (în timpul execuției). Biblioteca de clase MFC folosește informația de clasă runtime pentru a oferi informație de diagnosticare la detectarea erorilor în program sau la serializarea obiectelor în sau din spațiul de stocare.

□ *Clasa de bază CWnd*

Această clasă, este derivată din CObject și oferă un grad mare de funcționalitate, propriu tuturor ferestrelor dintr-un program MFC. Sunt incluse aici și casetele de dialog și controalele, care nu sunt decât versiuni specializate de ferestre. Clasa CWnd definește funcții care pot fi aplicate oricărui obiect din CWnd, inclusiv acelor obiecte care sunt exemplare ale claselor derivate din CWnd.

Clasele de bază CObject și CWnd sunt două exemple de clase de bază care se utilizează pentru a asigura funcțiile esențiale ale unui mare număr de clase prin intermediul unei clase de bază.

Aproape fiecare obiect semnificativ dintr-un program MFC este un exemplar din clasa CObject. Aceasta permite utilizarea suportului MFC pentru descoperirea multor pierderi frecvente de memorie și a altor tipuri de erori de programare. De asemenea clasa CObject declară funcții care pot fi utilizate pentru a oferi diagnosticări în timpul rulării și suport pentru serializare.

Fiecare fereastră dintr-un program MFC este un obiect CWnd. Clasa CWnd derivă din clasa CObject, deci are încorporate toate funcțiile acesteia. Utilizarea clasei CWnd pentru manevrarea controalelor și a ferestrelor din program permite programatorului să beneficieze de avantajele polimorfismului: clasa CWnd **asigură toate funcțiile generale de fereastră pentru toate tipurile de ferestre**. Aceasta înseamnă că în multe cazuri nici nu este necesară cunoașterea exactă a tipului controlului sau ferestrei accesate prin intermediul unui pointer CWnd.

Clasele CObject și CWnd sunt folosite în diverse moduri. Clasa CObject este folosită în mod normal drept clasă de bază atunci când programatorul își creează propriile clase. Clasa CWnd este adeseori transferată drept parametru de funcție și este folosită ca pointer generic pentru orice tip de fereastră într-un program MFC.

Când este folosită drept clasă de bază CObject asigură o mare parte din funcțiile de bază pentru o altă clasă. Se poate controla numărul acestor funcții asigurate de CObject utilizând funcții macro în declarația clasei derivate și în fișierele de definire.

Există patru nivele de asistență asigurate de clasa CObject claselor sale derivate:

- √ **nivelul de bază**, cu detecția pierderilor de memorie, nu necesită funcții macro;
- √ **asistența pentru identificarea claselor la rulare** necesită utilizarea funcției macro DECLARE_DYNAMIC în declarația clasei și a funcției IMPLEMENT_DYNAMIC în definiția clasei;
- √ **asistența pentru crearea dinamică a obiectelor** necesită utilizarea funcției macro DECLARE_DYNCREATE în declarația clasei și a funcției macro IMPLEMENT_DYNCREATE în definiția clasei;
- √ **asistența de serializare** necesită utilizarea funcției macro DECLARE_SERIAL în declarația clasei și a funcției macro IMPLEMENT_SERIAL în definiția clasei.

Toate funcțiile macro ale clasei CObject au un mod de utilizare asemănător. Toate funcțiile macro de tip DECLARE au un singur parametru, numele clasei. Funcțiile macro de tip

IMPLEMENT necesită, în general doi parametri: numele clasei și numele clasei de bază imediat superioare. Funcția IMPLEMENT_SERIAL reprezintă o excepție, deoarece necesită trei parametri.

Există două moduri de creare dinamică a obiectelor. Prima metodă, utilizează operatorul C++ new pentru a alocă dinamic un obiect din spațiul de alocare disponibil. O a doua metodă este folosită în special în cadrul MFC și folosește o clasă specială CRuntimeClass, și funcția macro RUNTIME_CLASS. Clasa CRuntimeClass se poate folosi pentru a **determina tipul unui obiect** sau pentru **crearea unui obiect nou**.

Biblioteca de clase MFC oferă mai multe facilități de diagnosticare, majoritatea aflându-se sub forma unor funcții macro folosite numai într-o versiune de depanare a programului, care oferă posibilitatea de a beneficia de avantajele ambelor situații (rulare și depanare). În faza de dezvoltare și de testare a programului, se pot folosi funcțiile de diagnosticare MFC pentru a verifica existența unui număr minim de erori, deși programul rulează cu o suprasolicitare indusă de operația de diagnosticare. Ulterior, când programul este compilat și devine o versiune utilizabilă, aceste verificări sunt eliminate și programul se execută la viteză maximă.

Există trei funcții macro utilizate frecvent într-un program MFC:

- ◆ **ASSERT** prezintă o casetă de dialog cu un mesaj de eroare când primește o expresie evaluată drept falsă, această funcție macro este compilată numai în variantele de depanare;
- ◆ **VERIFY** funcționează exact ca ASSERT, cu deosebirea că expresia evaluată este întotdeauna compilată, chiar și în variantele non-depanare, deși expresia nu este testată în versiunile finale;
- ◆ **ASSERT_VALID** testează un pointer către un exemplar din clasa CObject și verifică dacă obiectul este un pointer valid într-o stare validă. O clasă derivată din CObject poate invalida funcția ASSERT_VALID pentru a permite testarea stării unui obiect.

Funcțiile macro ASSERT și VERIFY sunt utilizate cu orice expresie, nu numai cu cele care implică CObject. Deși ambele efectuează teste pentru a se asigura că expresia evaluată are valoarea **TRUE**, există o diferență importantă între modurile de operare ale acestora. Când se compilează pentru o versiune finală, funcția ASSERT și expresia evaluată de aceasta sunt complet ignorate în timpul compilării. Și funcția macro VERIFY este ignorată, dar expresia este compilată și utilizată în versiunea finală.

O sursă comună de erori în programele MFC este *plasarea de coduri importante în interiorul unei funcții macro ASSERT în loc de VERIFY*. Dacă acea expresie este necesară pentru ca programul să lucreze corect, trebuie amplasată în interiorul unei funcții macro VERIFY și nu ASSERT.

În afara funcțiilor de diagnosticare și a celor macro, clasa CObject mai declară o **funcție virtuală care afișează conținutul unui obiect în timpul rulării**. Această funcție dump, este utilizată pentru a transmite mesaje cu privire la starea curentă a unui obiect către o fereastră a programului de depanare.

Dacă se depanează un program MFC, mesajele sunt afișate într-o fereastră de ieșire a programului de depanare. Se va adăuga declarației de clasă CObiectMeu codul sursă, funcția dump este plasată, de obicei în secțiunea de implementare a declarației de clasă. Deoarece dump este apelată numai de cadrul de lucru MFC, este de obicei declarată drept **protected** sau **private**. Deoarece funcția dump este apelată numai în variantele de depanare, declarația este înconjurată de instrucțiunile #ifdef și #endif, care elimină declarația funcției dump pentru versiunile finale.

Clasa CWnd este utilizată în mod normal ca un pointer generic de fereastră, și conține funcții suplimentare ce pot fi aplicate majorității ferestrelor. Cele mai des folosite sunt:

- ✓ **ShowWindow**, este folosită pentru ascunderea sau afișarea unei ferestre; funcția trebuie să aibă un parametru care să indice modul în care comanda influențează fereastra (există 10 parametri posibili cei mai frecvenți sunt SW_HIDE pentru ascunderea ferestrei și SW_SHOW, care afișează fereastra folosind dimensiunea și poziția curentă a acesteia);
- ✓ **SetWindowText** este utilizată pentru a stabili legenda sau bara de titlu a unei ferestre, dacă fereastra este un control de editare, se returnează conținutul controlului;

- ✓ **GetWindowText** returnează legenda sau bara de titlu a unei ferestre;
- ✓ **MoveWindow** este utilizată pentru a preciza noua poziție a unei ferestre sau a unui control;
- ✓ **SetFont** permite definirea corpului de literă utilizat pentru o fereastră, este valabilă pentru toate ferestrele, inclusiv pentru controalele de editare.
- ✓ **GetFont** returnează corpul de literă curent utilizat de o anumită fereastră.

□ *Clasa MFC CString*

Clasa **CString** reprezintă o implementare a obiectului șiruri tip matrice și a metodelor aferente acestuia. Această clasă, creează șiruri inteligente care pot fi cu ușurință adunate, copiate sau manevrate în alt mod, fără a avea nici una dintre problemele care apar la lucrul cu matricele de caractere.

Unul dintre marile avantaje ale clasei CString este faptul că un **obiect CString** se comportă exact ca unul dintre tipurile fundamentale, ca un întreg. Un obiect CString poate fi adăugat, atribuit sau copiat exact ca un întreg, însă totuși există anumite funcții membru definite pentru clasa CString.

Clasa CString oferă o funcție membru **GetLength** care returnează numărul de caractere stocate într-un obiect CString. Această funcție este utilă la stocarea de obiecte CString sau la pregătirea pentru diverse tipuri de date de ieșire. Pentru localizarea în interiorul unui obiect CString a unui caracter sau a unui șir de caractere se poate folosi funcția membru Find. Această funcție este utilă atunci când se caută caracterele de control în șirurile de intrare.

Funcția membru **SetAt** este utilizată pentru a transforma caracterul retur de car (CR) într-un spațiu. Funcția se poate folosi pentru a transforma orice caracter, atâta timp cât noul caracter nu este nul.

2.1.3.3. Clase tip colecție

O clasă tip colecție este utilizată pentru a crea obiecte capabile de a stoca alte obiecte, așa cum este cazul claselor container. Obiectele stocate pot aparține unuia dintre tipurile încorporate, cum ar fi int, char sau long, sau pot fi obiecte de tip CString. Există trei tipuri fundamentale de clase tip colecție conținute de biblioteca de clase MFC.

- ✚ **Matrice.** O colecție tip matrice permite stocarea și accesarea datelor utilizând operatorul indice sau []. Clasa de matrice este optimizată pentru a permite un acces facil la un anumit articol din matrice, dacă poziția sa este cunoscută. Totuși inserarea unui articol în mijlocul unei matrice poate fi foarte costisitoare sub aspectul timpului de procesare.
- ✚ **Liste.** O colecție tip listă își stochează articolele într-o listă tip lanț (chain-like list), ceea ce facilitează în mare măsură adăugarea sau eliminarea articolelor, indiferent de amplasarea acestora. Colecțiile tip listă au un cap și o coadă. Articolele pot fi inserate cu ușurință la capul sau la coada listei. Totuși, căutarea unui articol într-o listă mare poate fi dificilă deoarece articolele din listă trebuie testate pentru a găsi un anumit articol; în medie, aceasta înseamnă testarea a aproximativ jumătate din articolele unei liste la fiecare căutare.
- ✚ **Hărți.** O hartă este o colecție ceva mai complexă. Fiecărui articol stocat într-o hartă îi este asociată o cheie unică.

Cheia fiecărui articol este unică și este utilizată pentru accesul la articolul stocat în hartă. Adăugarea, eliminarea sau accesarea oricărui articol stocat într-o hartă este o operație foarte simplă; totuși este necesară cheia articolului care trebuie recuperat. Această cheie este convertită într-un index folosit la accesarea fiecărui articol în parte. Fiecare cheie este convertită mai întâi într-o valoare codificată (hash value), care este folosită apoi ca indice într-o matrice cu articolele colecției, cunoscută sub numele de **tabel de codificare** (hash table). Dimensiunea acestui tabel se poate specifica la crearea hărții; un tabel de mari dimensiuni impune spațiu de stocare mărit, în timp ce un tabel redus mărește posibilitatea ca două chei să genereze valori codificate egale, proces cunoscut sub numele de coliziune. Clasele MFC tip hartă tratează coliziunile prin crearea unor liste de articole care au chei egale.

Cele trei tipuri de colecție sunt disponibile în versiuni șablon și non șablon. Versiunile non-șablon, specifică un anumit tip de articol care urmează a fi stocat în colecție.

Clasele tip colecție MFC oferă o mare varietate de colecții care se pot folosi în programe MFC. Clasele tip colecție sunt furnizate ca parte a bibliotecii de clase, deci utilizarea acestora necesită foarte puțin efort de scriere din partea programatorului. Reutilizarea acestor clase permite economisirea a mii de linii de cod în aplicații.

Colecția CList

Este folosită adeseori atunci când o colecție de articole este utilizată ca un lanț de articole înrudite, de exemplu dacă se dorește accesarea articolelor de la începutul sau de la sfârșitul listei. Colecția CList permite adăugarea de articole la începutul și la sfârșitul listei. Pentru a insera un articol în capul unei liste din CList, se folosește funcția membru *AddHead*.

Pentru a traversa sau a examina fiecare articol stocat într-o colecție, este utilizată o variabilă **POSITION**. O variabilă **POSITION** este asemănătoare unui cursor într-un program de procesare a textelor. Prin apelarea funcției membru *GetHeadPosition* variabila **POSITION** este plasată exact anterior primului articol stocat în listă. Funcția membru *GetNext* execută două operații: returnează următorul articol stocat pe listă și avansează variabila **POSITION** către următorul articol din CList. Dacă această variabilă are valoarea **NULL**, atunci nu mai există articole în CList.

Toate clasele tip colecție dispun de o funcție *RemoveAll*, care este utilizată pentru a înlătura și a șterge toate articolele stocate într-o colecție.

Clasa tip colecție CArray

Clasa CArray este utilizată pentru a crea colecții care sunt mai utile decât matricele încorporate în C++. Clasa CArray asigură verificarea de domeniu, sub formă de apeluri ale funcției *ASSERT* la compilarea programului, cu activarea opțiunilor de depanare. De asemenea, permite dezvoltarea dinamică a matricei la adăugarea de elemente noi, aspect imposibil în cadrul matricelor încorporate.

Numărul de articole stocate în CArray poate fi furnizat prin apelarea funcției membru *GetSize*, accesul la diverse articole stocate în CArray poate fi realizat folosind operatorul de indexare (`[]`), ca și în cazul matricelor încorporate. Funcția *GetSize* se folosește pentru garantarea faptului că articolul respectiv există, iar dacă nu există certitudinea stocării unui anumit membru, se va folosi în schimb funcția *GetAt*.

Clasa tip colecție CMap

Clasa CMap este folosită pentru crearea colecțiilor care se indexează folosind o cheie unică pentru fiecare articol stocat în colecție. Fiecare articol stocat în colecție trebuie să aibă o cheie unică; dacă se încearcă stocarea a două articole cu aceeași cheie, primul articol va fi înlocuit de către al doilea articol.

Clasa tip colecție CMap este ideală pentru stocarea articolelor care dețin numere de identitate sau de serie unice. De exemplu, mărcile salariaților, seria împreună cu numărul buletinului de identitate sau indicii ISBN ai cărților reprezintă toate exemple bune de chei, deoarece sunt în mod garantat unice.

Lista de parametri ai șablonului CMap este mult mai complicată decât listele de parametrii folosite pentru CList și CArray. Primii doi parametri sunt folosiți pentru tipul de cheie. Primul parametru reprezintă tipul de cheie folosit ca argument în funcțiile membru.

Următorii doi parametri se referă la articolul stocat în colecție. Al treilea parametru reprezintă tipul de articol stocat în colecție, iar al patrulea parametru reprezintă tipul de articol folosit ca argument în funcțiile membru.

Funcția membru *SetAt* este folosită pentru stocarea unui articol și a unei chei în colecția CMap. Funcția membru CMap este adesea folosită pentru recuperarea unui articol din colecție, în funcție de o anumită cheie. În colecțiile de mari dimensiuni, CMap este cu mult mai rapidă decât CList sau CArray în materie de căutare, inserare sau ștergere a unui articol arbitrar, în

funcție de o cheie.

Căutarea într-o CMap este foarte asemănătoare cu procesul analog într-o CList. Funcția membru GetStartPosition returnează o variabilă **POSITION** care este avansată către articolul următor folosind funcția membru GetNextAssoc. Totuși, în afară de valoarea cheii, nu există nici o ordine a articolelor stocate într-o CMap. Dacă CMap este traversată folosind funcția GetNextAssoc, articolele din colecție sunt returnate în funcție de structura internă a CMap, și nu în funcție de ordinea în care au fost inserate.

UTILIZAREA BAZELOR DE DATE, PROGRAMAREA OLE ȘI COM

2.2.1. Utilizarea bazelor de date

Organizațiile cu profil economic acumulează o cantitate mare de informații în urma activităților comerciale de zi cu zi. Sunt înregistrate detalii despre clienți și furnizori, despre vânzări și necesarul de stocuri etc. Marea majoritate a corporațiilor optează pentru stocarea acestor date vitale în cadrul bazelor de date sau, mai exact, într-un **DMS (DataManagementSystem - sistem de gestionare a bazelor de date)**. Un sistem de gestionare a bazelor de date este un produs software care stochează datele într-o structură bine organizată și oferă mijloace eficiente de accesare și actualizare a acestor date.

Există două tipuri de baze de date: **baze de date relaționale** și **baze de date obiectuale**. Diferența dintre ele provine din modul conceptual în care sunt gestionate datele.

Bazele de date relaționale au la bază tipuri de date simple, recunoscute (caractere, șiruri, întregi etc.) și nu permit crearea unor noi tipuri de date.

Bazele de date obiectuale operează cu tipurile de date la un nivel mai înalt, permițând crearea prin definiție a unor noi tipuri de date. Aceste obiecte corespund celor create într-un limbaj orientat-obiect, de exemplu un obiect stocat într-o bază de date obiectuală ar putea fi o persoană sau un automobil.

Există mai mulți producători care oferă baze de date relaționale, fiecare bază de date are propria sa structură și, în consecință, un set propriu de funcții API. Din ce în ce mai mult se cere ca aplicațiile să poată lucra cu orice bază de date. Pentru a putea dezvolta aplicații care să utilizeze o bază de date relațională, indiferent de producătorul acesteia, este nevoie de o metodă generică de programare. O astfel de metodă există și se numește **ODBC (Open Database Connectivity - conectivitate deschisă a bazelor de date)**.

ODBC aduce o interfață de programare standard prin care pot fi utilizate diferite tipuri de baze de date relaționale. În acest scop este necesară existența unui intermediar care să traducă apelurile ODBC standard în apeluri de funcții specifice bazei de date. Acest intermediar este driverul ODBC, oferit de către producătorul bazei de date sau de către o firmă terță specializată în astfel de produse. ODBC a fost adoptat ca standard și astfel de drivere există acum pentru toate bazele de date răspândite.

Procedura de instalare a unor aplicații Microsoft, precum Office și Visual Studio, permite instalarea celor mai folosite drivere ODBC produse de Microsoft.

Comenzile sunt transmise driverului ODBC și pasate mai departe bazei de date cu ajutorul SQL (limbaj structurat de interogare). Acest limbaj a fost dezvoltat anume pentru accesarea bazelor de date și este acum standardul de facto. Există aproape tot atâtea variante de SQL câte baze de date sunt, fiecare producător aducând propriile îmbunătățiri. Dar există cerințe minime care asigură un set de comenzi care nu sunt standard, deoarece ODBC oferă o metodă de scurt-circuitare ce permite execuția directă a instrucțiunilor SQL. Utilizarea unor comenzi în afara standardului înseamnă ca aplicația să-și piardă capacitatea de a accesa bazele de date ale altor producători, tocmai această capacitate fiind avantajul major adus de ODBC.

Limbajul structurat de interogare **SQL** conține trei tipuri de comenzi: DDL, DML și DCL.

Comenzile DDL (Data Definition Language - limbaj pentru definirea datelor) sunt folosite *pentru crearea și modificarea schemelor de baze de date.*

Comenzile DML (Data Manipulation Language - limbaj pentru manipularea datelor) sunt folosite *pentru interogarea și manipularea informațiilor.*

Comenzile DCL (Data Control Language - limbaj pentru controlul datelor) sunt folosite pentru *acordarea unor drepturi de acces specifice diferiților utilizatori.*

Prima sarcină legată de utilizarea ODBC este configurarea unei surse de date. O sursă de date indică programului unde se află fișierele care conțin baza de date și ce driver ODBC trebuie folosit pentru interpretarea administratorului ODBC pentru sursele de date, accesibil din panoul de control.

2.2.2. Noțiuni de programare OLE și COM

Complexitatea crescândă a dezvoltării aplicațiilor în cadrul unor medii complexe, așa cum este Windows, cu multele sale interfețe de programare a aplicațiilor (API), precum și nevoia de standardizare, de control al versiunilor, de accelerare a dezvoltării și de calcul distribuit, au dus la apariția unei tehnologii numite **programare bazată pe componente**.

Următorul pas în evoluția COM va fi COM +, Microsoft promite că acesta va simplifica în bună măsură scrierea codului necesar pentru COM, astfel că obiectele COM vor avea un comportament mai similar cu obiectele C++ și vor putea fi create prin intermediul cuvintelor cheie **new** și **delete** din C++.

Componentele sunt entități software de dimensiuni mici (asemeni instanțelor unei clase) care efectuează operații specifice prin intermediul unor interfețe bine definite. Spre deosebire de instanțele claselor, componentele nu sunt legate definitiv de o instanță a unui program sau de un sistem gazdă, pot fi scrise într-o multitudine de limbaje, și cu toate acestea, comunică fără probleme, prin intermediul interfețelor, cu programe și componente implementate în alte limbaje.

Microsoft a dezvoltat această tehnologie până la forma actuală, fiind denumită în prezent **Component Object Model (COM)** - modelul componentelor obiectuale. Se pot întâlni și alți temeni înrudiți, precum **legarea și încapsularea obiectelor (Object Linking and Embedding-OLE)** sau **controale ActiveX**. Acestea reprezintă implementări particulare ale programării COM. Propriu-zis COM este un standard independent de limbaj și de platformă care definește modul în care obiectele pot comunica între ele prin intermediul unui protocol acceptat în comun.

Cel mai important element privind obiectele COM îl reprezintă **interfețele** acestora; obiectele propriu-zise nu sunt decât cutii negre care implementează o funcționalitate particulară. Pentru a putea să se utilizeze această funcționalitate, programele trebuie să respecte un contract bine definit privind transmiterea parametrilor și obținerea rezultatelor. Acest contract dintre programul client și obiect se numește **interfață**.

Întregi biblioteci **API** pot fi definite și proiectate în termeni de interfață. Dacă se dezvoltă o aplicație client, se poate scrie programul astfel încât să comunice cu un server prin intermediul acestor interfețe acceptate, fără a mai conta ce aplicație server va fi utilizată. Reciproc, se poate decide dezvoltarea unor componente de tip server care respectă definițiile interfețelor, comercializându-le ca alternativă viabilă la componentele altor producători.

Interfața de programare a aplicațiilor pentru mesagerie reprezintă un set de interfețe standard pentru obiectele COM. Oricine este liber să scrie obiecte COM care efectuează operații precum *stocarea mesajelor, transportul mesajelor și oferirea către destinatarii mesajelor a unei agende de adrese.*

Microsoft Exchange este o implementare particulară a acestor componente server, dar există multe alte implementări. Codul efectiv s-ar putea să difere enorm între diferite implementări, dar toate obiectele **COM** respectă aceleași specificații de interfață. Aceasta înseamnă că, toți clienții acestor servicii, Microsoft Outlook, pot folosi componentele compatibile **MAPI** ale oricărui producător în scopul trimiterii, primirii și stocării mesajelor. În

mod similar, orice producător poate să ofere propriile programe client (și mulți o fac) care folosesc Exchange sau orice alte componente server MAPI, chiar fără a ști ale cui sunt componentele utilizate. Singura cerință în ceea ce privește componentele client și server este ca acestea să se apeleze reciproc prin intermediul acestor interfețe definite de comun acord, reunite sub numele de MAPI.

Interfețe COM

O interfață este o definiție a unei mulțimi de funcții și a parametrilor acestora. Orice obiect COM dispune de cel puțin o interfață, iar în mod frecvent sunt oferite mai multe interfețe, fiecare conținând propriul său set unic de funcții.

Un obiect COM poate fi scris în orice limbaj care are capacitatea de a suporta aceste interfețe. Unele limbaje sunt mai potrivite în acest sens decât altele, de exemplu JAVA este foarte potrivit, din cauză că, fiecare obiect JAVA poate avea mai multe interfețe, deci se mapează natural peste un obiect COM. Obiectele COM conțin codul efectiv aflat în spatele acestor interfețe, astfel că un program care apelează o funcție declarată într-o interfață va găsi o implementare ce efectuează operația definită de acea funcție în cadrul interfeței respective.

Structura interfeței COM standard este exact aceeași cu structura tabelii de funcții virtuale din Visual C++, ceea ce înseamnă că este posibil să se folosească mecanismul tabelilor de funcții virtuale pentru a defini și implementa interfețe COM.

În mod normal, funcțiile virtuale sunt folosite ca o metodă de supradefinire în cadrul unei clase derivare a unei funcții din clasa de bază. În acest scop este declarată o tabelă de funcții virtuale asociată clasei respective.

Orice instanță de memorie a unui obiect C++ are atașată o tabelă de funcții virtuale (chiar dacă se poate ca unele tabele să nu conțină nici o intrare și deci, să fie vide). Fiecare intrare din tabelă conține un pointer către codul care implementează o funcție virtuală. De fiecare dată când se apelează una dintre funcțiile virtuale ale obiectului, tabela atașată oferă adresa corectă, corespunzătoare fie funcției din clasa de bază, fie funcției din clasa derivată.

Declararea unei clase C++ care conține exclusiv funcții virtuale pure, se numește **clasă de bază abstractă**. Nu este posibilă instanțierea unei clase abstracte, dar aceste clase se pot utiliza pentru crearea unei definiții de interfață COM compatibilă cu C++. La crearea unei instanțe a unui obiect COM, un proces server aflat în rulare se va ocupa de inițializarea și gestionarea acesteia. Procesul server poate fi propriul program client, un DLL atașat, un executabil (.exe) aflat pe propriul calculator sau un program aflat pe un sistem la distanță. Fiecare din aceste ipostaze diferite se numește **context**.

Când se apelează funcții pe baza acestui pointer la interfață, propriul client și obiectul COM sunt puse în legătură cu ajutorul unui DLL terț, care se numește **de intermediere**, tehnica purtând numele de **apel intermediat**. Biblioteca de intermediere are în sarcină *împachetarea parametrilor într-un format independent de mașină, în scopul transmisiei*. Este posibil apoi, să folosească apelurile de procedură la distanță pentru a apela o funcție a unui obiect COM aflat la distanță. Bibliotecile de intermediere pot fi generate automat, prin crearea definițiilor în limbajul de definire a interfețelor (IDL) și prelucrarea acestora de către compilatorul Microsoft IDL.

Vechea problemă a întreținerii versiunii corecte a aplicației este rezolvată printr-o simplă regulă. Odată ce s-a eliberat un obiect COM pentru a fi folosit de utilizatorii de pretutindeni, versiunile ulterioare trebuie să păstreze ne-alterate interfețele originale. Aceasta înseamnă că noile versiuni ale unui obiect COM trebuie să implementeze interfețe suplimentare, fără a le modifica pe cele originale. Programele client mai vechi, care nu știu să utilizeze decât interfața mai veche, vor solicita în continuare această interfață, în timp ce programele client care cunosc noile facilități pot să solicite versiunea mai nouă de interfață.

Automatizarea OLE

Legarea și încapsularea obiectelor (OLE) este un termen folosit inițial pentru a *descrie capacitatea de a insera obiecte de diferite tipuri în documente având un tip propriu*, un exemplu

fiind inserarea foilor de calcul Excel în documentele Word.

Documentele care suportă operațiile de legare și încapsulare se numesc **documente compuse**. Obiectele inserate pot fi încapsulate, ceea ce înseamnă că pot fi salvate odată cu documentul, sau legate, ceea ce înseamnă că în document va fi inserată o referință la alt fișier (numită identificator). Acestea sunt cele două operații care au dat naștere termenului OLE original.

Semnificația acestui termen, s-a lărgit însă, incluzând acum **tragerea și plasarea OLE și automatizarea OLE**, acestea din urmă referindu-se la situația în care *un program poate să apeleze metodele unui alt program care rulează ascuns, fără ca utilizatorul să fie conștient de această interacțiune*.

Un obiect COM care oferă o interfață dispecer conține o tabelă de metode (funcții), evenimente (funcții care utilizează apeluri inverse ale clientului în scopul unei notificări a acestuia) și proprietăți (funcții care furnizează sau stabilesc valoarea unei anumite variabile a obiectului COM). Programele client pot să acceseze aceste informații dinamic (procedeul se numește legare târzie), în timpul rulării, pentru a afla detalii privind obiectul de automatizare.

Se poate întâlni termenul de interfață duală folosit în legătură cu COM și OLE. Un obiect COM poate să implementeze o interfață duală, oferind funcțiile atât printr-o interfață directă COM, cât și printr-o interfață dispecer. Astfel, programele client pot beneficia de avantajele ambelor soluții; programele C++ rapide pot să acceseze obiectul direct prin interfața COM rapidă, iar Visual Basic și limbajele de scriptare pot să acceseze funcțiile prin interfața dispecer mai înceată.

✚ Crearea unui server propriu de automatizare OLE

Multe aplicații, printre care Word, Excel sau chiar Developer Studio, sunt **servere de automatizare**. Acestea pun la dispoziția aplicațiilor client o interfață dispecer care poate fi utilizată pentru apelarea funcțiilor proprii, oferind servicii specializate precum verificarea ortografică.

Visual Basic Scripting, instrumentul utilizat pentru crearea macro-comenzilor, folosește aceste metode în scopul creării și manipulării documentelor din cadrul serverelor de automatizare. De fiecare dată când se scrie o macro-comandă pentru unul din aceste programe, se folosește o versiune redusă de Visual Basic care permite apelarea funcțiilor din interfața dispecer a serverului de automatizare respectiv.

Serverele de automatizare sunt asociate de obicei cu un nume inteligibil care poate fi utilizat pentru determinarea identificatorilor de clasă corespunzători, care sunt reținute într-un registru intern, conținând subchei ce specifică identificatorul de clasă al serverului de automatizare respectiv.

Infrastructura unei aplicații server de automatizare poate fi creată prin intermediul AppWizard, validând opțiunea **Automation** din pasul trei din MFC AppWizard.

✚ Servere și containere OLE

Un **server OLE** este o aplicație care poate fi lansată în cadrul unei ferestre a unei aplicații container. De exemplu, foile de calcul Excel, pot fi inserate și editate în Word. În acest caz, Word reprezintă aplicația container OLE, iar Excel este aplicația server OLE.

Se mai întâlnește termenul de **document activ**, folosit în legătură cu serverele și containerele OLE, ceea ce este o terminologie relativ nouă, legate de ActiveX, care dezvoltă **arhitectura container/server OLE** pentru a permite obiectelor încapsulate să câștige controlul asupra întregii zone client a containerului (nu doar asupra unui mic cadru) și să manipuleze direct fereastra, meniurile și barele cu instrumente ale acestuia.

Aplicațiile pot fi concomitent containere OLE și servere OLE, caz în care pot juca ambele roluri. Word-ul și Excel-ul sunt exemple potrivite în acest sens, deoarece se pot rula Excel și să se insereze documente Word și viceversa.

Un server complet poate să ruleze ca aplicație de sine stătătoare sau ca obiect inserat, în

timp ce mini-serverele pot fi lansate doar din cadrul unui program container. La inserarea unui obiect server OLE, acesta poate fi editat în interiorul unei suprafețe rectangulare care este mărginită de un chenar gros, hașurat, de culoare neagră, numit cadru local. Operația se numește **activare locală** și este inițiată prin transmiterea unor indicatori, numiți verbe, de la container către server. Atunci când nu este activ, cadrul dispare și imaginea afișată este ultima generată atunci când acesta era activ (stocată și reprodusă pe baza unui context dispozitiv de tip metafîșier). Atunci când este activ, cadrul este afișat, iar la meniul propriu al container-ului sunt adăugate elemente de meniu ale obiectului server.

Datele documentului încapsulat pot fi serializate de către documentul container prin intermediul funcțiilor obișnuite de serializare. Dacă un document încapsulat este stocat integral în cadrul documentului compus al aplicației client, obiectele legate sunt stocate sub forma unor simpli identificatori. Un astfel de identificator este un mic obiect care identifică în mod unic locația datelor propriu-zise și modul de afișare a acestora în cadrul aplicației client.

Infrastructura standard pentru **servere OLE** și **containere OLE** poate fi creată cu ajutorul AppWizard, însă cu toate acestea, între container și server există o cooperare complexă care trebuie implementată pentru a asigura funcționarea corespunzătoare a ambelor părți. Prin personalizarea a două clase adăugate pe partea de server și a clasei suplimentare a containerului se poate implementa suport pentru operații destul de complexe de editare locală la rularea într-un cadru în fereastra unei aplicații container. Nici serverul și nici containerul nu au voie să cunoască tipul obiectului care încapsulează sau al celui conținut. Atât timp cât obiectele server și client respectă același standard, containerul și serverul pot fi integrate perfect pentru a da utilizatorului impresia unei aplicații unitare.

2.3. TIPURI SI STRUCTURI DE DATE. TIPURI DE DATE ABSTRACTE

Studiul datelor cuprinde următoarele probleme:

- mașini pentru stocarea datelor
- limbaje de descriere a manipularii datelor
- fundamente care descriu ce date putem obține prelucrând date primare
- structuri de reprezentare a datelor

Obiectivele studiului structurilor de date sunt următoarele:

- explorarea diverselor modalități de structurare a datelor
- identificarea clasei de operații care pot executate asupra fiecărei structuri
- modul de reprezentare în calculator al obiectelor aparținând structurii respective astfel încât aceste operații să poată fi executate cât mai eficient

Scopul studiului structurilor de date este reprezentat de însușirea și stăpânirea următoarelor două tehnici fundamentale:

- proiectarea unor reprezentări adecvate ale datelor
- proiectarea și analiza algoritmilor care operează cu aceste reprezentări

Orice constantă, variabilă, expresie sau funcție se caracterizează printr-un anumit tip de date. Prin tip de date se înțelege o mulțime de elemente numite valori sau obiecte, care pot clar distinse între ele. Într-un limbaj de programare, orice constantă aparține unui anumit tip de date, orice variabilă poate stoca valori de un anumit tip, orice expresie se evaluează la o valoare de un anumit tip și orice funcție întoarce o valoare de un anumit tip.

După modul de definire în raport cu un anumit limbaj de programare, tipurile de date se clasifică în:

- tipuri predefinite - limbajul permite folosirea variabilelor de tipul respectiv și furnizează o mulțime de operații predefinite pentru manipularea acestor variabile în cadrul unui program.
- tipuri definite de utilizator

Obiectele unui limbaj de programare sunt:

- constante: tipul rezultă din forma sintactică (modul de scriere) a lor
- variabile: tipul rezultă prin specificarea în program a unui enunț special – declarație
- expresii: tipul rezultă din tipurile operanzilor și modul lor de compunere cu ajutorul operatorilor.
- funcții: tipul rezultă din declarația funcției (în C - prototip sau semnătură).

Declarația este un enunț care asociază o variabilă sau o funcție cu tipul său. Definiția este un enunț care asociază o variabilă sau funcție cu tipul său și alocă spațiu de memorie variabilei sau corpului funcției respective.

Dupa modul de definire în termenii altor tipuri de date, tipurile de date se clasifică în:

- tipuri primitive
- tipuri derivate

Un obiect poate fi:

- compus: se poate descompune în mai multe componente
- simplu sau scalar.

La rândul său, un tip de date poate fi:

- compus: valorile sale sunt compuse
- simplu sau scalar

Un tip de date este ordonat dacă pe mulțimea valorilor sale se poate defini o relație de ordine. În caz contrar, se spune că este neordonat. Un tip de date este ordinal dacă pe mulțimea elementelor sale se poate defini o relație de ordonare liniară (este posibilă definirea predecesorului și succesivului unui element). Un tip ordinal modelează o mulțime finită sau numărabilă.

Cardinalitatea unui tip T reprezintă numărul de valori distincte aparținând tipului T.

O structură de date se definește ca mulțimea de obiecte care alcătuiesc un anumit tip, proprietățile obiectelor, relațiile dintre obiecte și operațiile asupra obiectelor. Definiția unei structuri de date trebuie să conțină trei elemente:

- numele structurii
- operațiile asupra structurii
- proprietățile structurii

Operațiile de bază aplicabile obiectelor unei structuri de date sunt:

- constructori – se referă la modul de generare a noi obiecte aparținând structurii respective;
- destructori – se utilizează atunci când gestiunea obiectelor se face explicit de către programator; eliberează memoria alocată obiectelor;
- selectori - se referă la modul de accesare a elementelor care compun un obiect compus;
- modificatori – se referă la modul de actualizare a elementelor care compun un obiect compus;
- recunoscători – se utilizează pentru recunoașterea diverselor proprietăți ale obiectelor aparținând structurii respective;
- testori – se utilizează pentru compararea obiectelor aparținând structurii respective

Proprietățile structurii nu descriu modul în care trebuie implementate operațiile structurii. Cu toate acestea, proprietățile unei structuri de date exprimă maniera în care trebuie să funcționeze operațiile structurii, indiferent de modul lor de implementare. Cu alte cuvinte, proprietățile abstractizează operațiile de modul lor de implementare. Din acest motiv, noțiunea de structură de date este cunoscută și sub numele de **tip abstract de date**, iar această manieră de definire a structurilor de date se numește abstractizarea datelor.

Formal, o structură de date poate fi definită ca un tuplu $S = \{D, d, O, P\}$, unde:

- D este o mulțime de tipuri de date implicate în definirea lui S
- $d \in D$ este tipul de date al obiectelor structurii S
- O este mulțimea operațiilor definite pe structura S
- P este mulțimea proprietăților structurii S

O implementare a unei structuri de date S este o transformare care definește modul de reprezentare a obiectelor aparținând tipului de date al structurii în funcție de obiectele altei structuri de date T . Fiecare operație a lui S trebuie definită în termenii operațiilor lui T

Cele mai întâlnite tipuri primitive în limbajele moderne de programare sunt:

a) **tipuri numerice**: ele sunt reprezentări ale unor mulțimi de numere cunoscute din matematică, precum N , Z sau R . Vom avea în consecință date întregi fără semn, întregi cu semn sau reali. Numerele întregi se reprezintă intern în calculator sub forma unei succesiuni de biți, cu ajutorul codului complement (cod complement față de 2). Numerele reale se reprezintă intern în calculator cu ajutorul reprezentării în virgulă mobilă (flotantă), care cuprinde semnul, mantisa și exponentul. Pentru a reprezenta corect și numerele subunitare, se utilizează reprezentarea cu semn, caracteristică și exponent.

b) **tipul caracter**: este destinat reprezentării caracterelor alfanumerice. Tipul caracter modelează mulțimea finită a tuturor caracterelor afișabile. Din nefericire, nu există un set standard de caractere specific tuturor sistemelor de calcul. Din acest motiv, termenul "standard" trebuie interpretat în acest context ca referindu-se la sistemul de calcul particular pe care este executat programul în cauză. Cel mai folosit set "standard" de caractere este cel definit de Organizația Internațională pentru Standarde (ISO) și anume versiunea sa americană (codul ASCII), care constă din 128 de caractere, dintre care primele 33 sunt caractere de control și celelalte 95 sunt caractere afișabile. Caracterele sunt codificate intern prin valori întregi numite coduri. Deoarece codul ASCII are 128 de caractere, pentru codificarea caracterelor sale sunt suficienți 7 biți. Cu toate acestea se folosește pentru codificare un octet, deoarece memoria internă a sistemelor de calcul este structurată de obicei sub forma unei secvențe de locații cu dimensiunea de un octet. Pentru seturile cu peste 256 de caractere, a fost introdusă specificația standard Unicode (cum este cazul limbii chineze sau japoneze), pentru reprezentarea căreia se folosesc doi octeți.

c) **tipul logic**: este destinat reprezentării mulțimii valorilor logice fals și adevărat.

d) **tipul enumerare**: este destinat reprezentării unei mulțimi finite de valori, fiecare valoare fiind desemnată printr-un nume simbolic. Se utilizează de obicei atunci când este necesară definirea unui tip de date prin indicarea explicită a elementelor sale. Elementele unui tip enumerare se reprezintă intern prin codificare cu ajutorul unor constante întregi.

e) **tipul subdomeniu**: este destinat reprezentării unui interval al unui tip ordonat liniar. Tipurile subdomeniu se utilizează de obicei pentru reprezentarea indicilor tablourilor.

f) **tipul referință**: este destinat reprezentării adreselor altor obiecte. Tipul referință se utilizează de obicei pentru implementarea unor obiecte abstracte complexe cum sunt listele și arborii. Numeroase prelucrări se referă la relațiile dintre obiectele prelucrate și nu doar la valorile lor. În astfel de cazuri poate fi necesară reprezentarea explicită a acestor relații. Pentru aceasta anumite obiecte trebuie să se poată referi la alte obiecte. Este posibil chiar ca un același obiect să fie referit din două sau mai multe locuri. Întrucât copierea valorii obiectului în locul de unde a fost referit nu este o soluție acceptabil din cauza consumului mare de memorie și dificultăților de menținere a consistenței, rezultă că trebuie apelat la o altă tehnică și anume de a defini în program obiecte care să poată referi alte obiecte. Pentru aceasta se folosesc tipul referință și obiectele pointer. O referință la un obiect se implementează ca adresă a locației de memorie, corespunzătoare obiectului respectiv.

Fiind dat un obiect o de tip T , el poate fi referit utilizând un obiect de tipul referință la T . Vom nota acest tip prin $\text{Ref}(T)$, definiția sa axiomatică fiind ilustrată în figura următoare:

structura $Ref(\mathcal{T})$

operații

$vid : \rightarrow Ref(\mathcal{T})$

$ref : \mathcal{T} \rightarrow Ref(\mathcal{T})$

$deref : \{X \in Ref(\mathcal{T}) : X \neq vid\} \rightarrow \mathcal{T}$

proprietăți

$(ref(X) = vid) = fals$

$ref(deref(P)) = P$

$deref(ref(X)) = X$

Constanta vid desemnează valoarea unui pointer care nu se referă la nici un alt obiect (o referință vidă).

Tipul referință la T se reprezintă în limbaj pseudocod prin $ref(T)$, ca în exemplul următor:

```
var int x, y
var ref int px
x ← 10
px ← ref(x)
y ← deref(px)
```

În urma executării acestei secvențe de instrucțiuni, variabila y primește aceeași valoare ca și x , adică 10, deoarece px are ca valoare o referință la x .

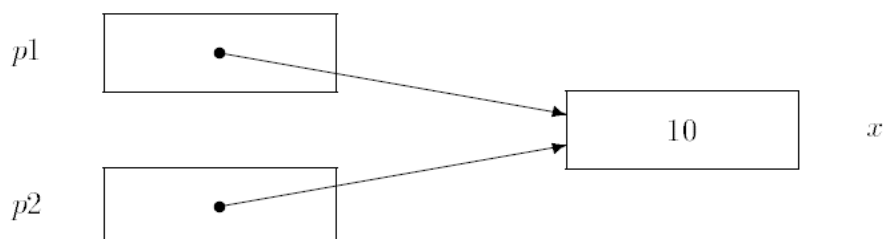
Un alt exemplu care realizează în mod diferit aceeași prelucrare este următorul:

```
var int x, y
var ref int px, py
x ← 10
px ← ref(x)
py ← ref(y)
deref(px) ← deref(py)
```

Se obișnuiește ca "legăturile" create între obiecte prin intermediul pointerilor să se reprezinte grafic. Spre exemplu, efectul executării secvenței:

```
var int x
var ref int p1, p2
x ← 10
p1 ← ref(x)
p2 ← ref(x)
```

se poate reprezenta grafic astfel:



O referință la un obiect se implementează ca adresă a locației de memorie, corespunzătoare obiectului respectiv. Operația *ref* devine astfel operația de încărcare a adresei unui obiect. Operația *deref* se implementează utilizând tehnica adresării indirecte, prin care o anumită valoare este interpretată drept adresa unei locații de memorie de unde se va extrage valoarea propriu-zisă.

Alocare statică și alocare dinamică

În cele discutate până acum, singura modalitate de a crea obiecte a constituit-o declararea acestora. O declarație are ca efect alocarea memoriei pentru obiectul respectiv la momentul compilării programului, obiectul fiind creat la încărcarea programului, deci înainte de execuția sa efectivă. Această modalitate de alocare a memoriei se numește statică. Principala caracteristică a acestui mod de alocare este faptul că obiectele sunt referite în program prin identificatorul introdus la declararea lor. Acest lucru este posibil deoarece asocierea dintre identificator și locația de memorie rezervată obiectului s-a făcut în faza de compilare a programului.

Obiectele pot fi însă create și la momentul execuției. Această modalitate de alocare a memoriei se numește alocare dinamică. Un astfel de obiect nu va putea fi însă accesibil prin program utilizând un identificator, deoarece momentul execuției succede momentului compilării programului. Din acest motiv, un obiect creat dinamic se numește și obiect anonim.

O modalitate elegantă de a accesa obiectele create dinamic o constituie utilizarea tipului referință. Pentru aceasta se introduce operatorul *nou* care primește un tip T și creează un obiect aparținând lui T , întorcând o referință la T . Fie τ mulțimea tuturor tipurilor și Ref mulțimea tuturor referințelor. Semnătura acestui operator este $nou : \tau \rightarrow Ref$.

Dacă operatorul nou întoarce referința vidă *vid*, atunci înseamnă că obiectul nu a putut fi creat. Dacă valoarea întoarsă este $p \neq vid$, atunci obiectul a fost creat și el poate fi accesat utilizând "numele" *deref(p)*.

Operația inversă alocării dinamice a unui obiect este eliberarea memoriei alocate. Această operație se poate realiza automat, atunci când obiectul respectiv nu mai este referit în program, sau manual de către programator prin apelul operatorului standard *elib* : $Ref \rightarrow$. Un exemplu de folosire este următorul:

```
var int x, y
var ref int p
...
p ← nou(int)
deref(p) ← x
x ← y
y ← deref(p)
elib(p)
```

Tipuri de date compuse

Din punct de vedere conceptual, orice obiect compus este un n-tuplu $o = (o_1, o_2, \dots, o_n)$, de obiecte o_i , $1 \leq i \leq n$, fiecare obiect o_i având tipul T_i . Tipul lui o este tipul compus $T = (T_1, \dots, T_n)$. Fiecare dintre tipurile T_i poate fi la rândul său primitiv sau compus.

Pentru orice tip compus, pe lângă operațiile fundamentale de atribuire și test de egalitate/inegalitate, se mai definesc de obicei următoarele două:

- operatorul constructor, care construiește un obiect compus $o = (o_1, o_2, \dots, o_n)$, din obiectele componente o_i

- operatorul de selecție, care permite accesul atât în citire cât și în scriere la fiecare dintre componentele unui obiect compus

Cele mai des utilizate tipuri de date compuse predefinite de majoritatea limbajelor de programare sunt:

- **tipul agregat**

Un obiect agregat, numit și înregistrare sau structură, este un obiect compus ale cărui componente (numite și câmpuri) sunt eterogene (au tipuri diferite) și au fiecare asociat câte un nume simbolic, folosit în cadrul operației de selecție.

Cardinalitatea tipului compus agregat este egală cu produsul cardinalităților tipurilor sale componente. La nivelul implementării, unui obiect agregat îi corespunde o zonă compactă de memorie a cărei mărime rezultă din mărimile componentelor obiectului respectiv – suma acestora.

Orice referință la un câmp al unui obiect agregat se transformă în distanța zonei alocate acestuia față de începutul zonei alocate obiectului. Distanțele se calculează de regulă la momentul compilării programului. Accesarea unui câmp al un agregat se implementează de regulă utilizând tehnica adresării relative.

structura $Agregat(\mathcal{S}, \tau, \mathcal{T})$

operații

$(\dots) : \times_{T \in \tau} T \rightarrow Agregat(\mathcal{S}, \tau, \mathcal{T})$

$set(S) : Agregat(\mathcal{S}, \tau, \mathcal{T}) \times \mathcal{T}(S) \rightarrow Agregat(\mathcal{S}, \tau, \mathcal{T})$

$get(S) : Agregat(\mathcal{S}, \tau, \mathcal{T}) \rightarrow \mathcal{T}(S)$

proprietăți

$get(S_1)(set(S_2)(A, T)) = (S_1 = S_2 : T ? get(S_1)(A))$

Reprezentarea axiomatică a tipului Agregat

- **tipul tablou**

Deseori este necesară folosirea unui tip compus ale cărui componente sunt toate de același tip. Un astfel de tip se numește omogen. În plus, accesul la componente are loc prin selectori care sunt elemente ale unei mulțimi finite și ordonate liniar. În aceste situații se va folosi tipul tablou.

Tipul tablou se definește pornind de la un tip de bază T care modelează tipul componentelor și de la un tip subdomeniu I , care modelează mulțimea selectorilor, numiți în acest caz indecși. Cardinalitatea tipului tablou se determină ridicând cardinalitatea tipului de bază la puterea cardinalității tipului indecșilor.

structura $Tablou(\mathcal{T}, \mathcal{I})$

operații

$(\dots) : \times_{I \in \mathcal{I}} \mathcal{T} \rightarrow Tablou(\mathcal{T}, \mathcal{I})$

$set : Tablou(\mathcal{T}, \mathcal{I}) \times \mathcal{I} \times \mathcal{T} \rightarrow Tablou(\mathcal{T}, \mathcal{I})$

$get : Tablou(\mathcal{T}, \mathcal{I}) \times \mathcal{I} \rightarrow \mathcal{T}$

proprietăți

$get(set(T, I, X), J) = (I = J : X ? get(J, T))$

Reprezentarea axiomatică a tipului Tablou

La nivelul implementării, unui tablou îi corespunde o zonă contiguă de memorie, caracterizată prin adresă de început și mărime. Să presupunem că min și respectiv max sunt limitele inferioară și respectiv superioară ale indicilor tabloului T și să presupunem de asemenea că pentru reprezentarea unui element al tabloului, aparținând tipului de bază T , sunt necesare n cuvinte de memorie. Rezultă că zona de memorie alocată tabloului va avea $(max-min+1) \times n$ celule, iar zona în care este memorat elementul $T[I]$ va începe la adresa $A = B + (I - min) \times n$, unde B este adresa de început a zonei alocate tabloului.

Accesarea unui element al un tablou se implementează de regulă utilizând tehnica adresării indexate.

- **tipul mulțime**

Numeroși algoritmi prelucrează colecții de obiecte în care ordinea elementelor nu contează. În astfel de situații se recomandă folosirea tipului mulțime.

Tipul mulțime se definește pornind de la un tip de bază T și el reprezintă toate submulțimile lui T . Un astfel de tip se mai numește și mulțimea putere a lui T .

Pentru reprezentarea internă a obiectelor de tip mulțime se folosește de obicei funcția caracteristică.

structura 2^T

operații

$$\emptyset : \rightarrow 2^T$$

$$inset : 2^T \times T \rightarrow 2^T$$

$$elim : 2^T \times T \rightarrow 2^T$$

$$\cdot \in \cdot : T \times 2^T \rightarrow Logic$$

$$\cdot \cap \cdot : 2^T \times 2^T \rightarrow 2^T$$

$$\cdot \cup \cdot : 2^T \times 2^T \rightarrow 2^T$$

$$\cdot \setminus \cdot : 2^T \times 2^T \rightarrow 2^T$$

proprietăți

$$(X \in \emptyset) = fals$$

$$(Y \in inset(M, X)) = (X = Y : adevarat ? Y \in M)$$

$$(elim(inset(M, X), Y) = (X = Y : elim(M, Y) ? inset(elim(M, Y), X))$$

$$elim(\emptyset, X) = \emptyset$$

$$inset(inset(Y, M), X) = (X = Y : inset(M, X) ? inset(inset(M, X), Y))$$

$$(\emptyset \cap M) = \emptyset$$

$$(inset(M, X) \cap N) = (X \in N : inset(M \cap N, X) ? M \cap N)$$

$$(\emptyset \cup M) = M$$

$$(inset(M, X) \cup N) = inset(M \cup N, X)$$

$$(\emptyset \setminus M) = \emptyset$$

$$(inset(M, X) \setminus N) = (X \in N : M \setminus N ? inset(M \setminus N, X))$$

Reprezentarea axiomatică a tipului mulțime

- **tipuri cu variante**

În practică este uneori convenabil să se considere că mai multe tipuri diferite sunt variante ale aceluiași tip. În acest caz se va defini un tip cu variante a cărei mulțime de valori este egală cu reuniunea disjunctă a tipurilor inițiale. Cardinalitatea unui tip cu variante se determină ca sumă a cardinalităților tipurilor variante ale sale.

2.4. LISTE, STIVE, COZI

Una dintre cele mai simple și mai des folosite structuri de date este lista liniară, cunoscută și sub numele de listă ordonată sau secvență. O listă liniară constă dintr-o înșiruire de n elemente X_1, X_2, \dots, X_n , numite noduri, aparținând de obicei unui același tip de bază T . Proprietățile structurale esențiale ale acestui șir se rezumă la poziția relativă a elementelor, așa cum apar ele în cadrul șirului, adică:

- dacă $n > 0$ atunci X_1 este primul nod și X_n este ultimul nod
- dacă $1 < i < n$ atunci nodul al i -lea, X_i , este precedat de nodul X_{i-1} și urmat de nodul X_{i+1} .

Ultima proprietate exprimă esența accesului secvențial, care afirmă că dacă se cunoaște nodul curent X_i atunci se poate accesa eficient nodul predecesor X_{i-1} și nodul succesori X_{i+1} .

Există o mare varietate de operații ce pot fi realizate asupra listelor liniare. Acestea includ:

- determinarea lungimii – a numărului de elemente - unei liste
- determinarea valorii primului, respectiv ultimului element al unei liste
- descompunerea unei liste în primul element și lista formată din restul elementelor, respectiv în ultimul element și lista restului elementelor
- adăugarea unui element la începutul, respectiv sfârșitul unei liste
- testarea dacă lista este sau nu vidă
- parcurgerea listei de la început la sfârșit sau invers
- determinarea valorii celui de al i -lea element al listei
- modificarea valorii celui de al i -ea element al listei
- inserarea unui element pe poziția i
- ștergerea elementului din poziția i

Alegerea setului corespunzător de operații depinde de aplicația în care sunt folosite listele. Restrângerea setului de operații ne conduce la obținerea unor noi structuri de date gen listă, dintre care unele apar foarte frecvent în aplicații și din acest motiv au căpătat nume speciale:

- stiva este o listă liniară pentru care toate inserările și ștergerile și în general orice tip de accese au loc numai la unul dintre capetele sale. Acesta se numește vârful stivei, iar celălalt capăt inaccesibil în mod direct se numește baza stivei
- coada este o listă liniară în care toate inserările se fac la unul dintre capete, numit spate, iar ștergerile și citirile au loc la celălalt capăt numit față cozii.

Listele pot fi definite recursiv astfel: o listă este fie lista vidă, fie este formată din elementul din capul listei - primul element al listei - și lista formată din restul elementelor. Aceasta definiție ne conduce la definiția axiomatică din figura următoare. Am notat cu T tipul elementelor listei.

structura $Lista(\mathcal{T})$

operații

$$vid : \rightarrow Lista(\mathcal{T})$$

$$cons : \mathcal{T} \times Lista(\mathcal{T}) \rightarrow Lista(\mathcal{T})$$

$$cap : \{L \mid L \in Lista(\mathcal{T}), \neg evida(L)\} \rightarrow \mathcal{T}$$

$$rest : \{L \mid L \in Lista(\mathcal{T}), \neg evida(L)\} \rightarrow Lista(\mathcal{T})$$

$$evida : Lista(\mathcal{T}) \rightarrow Logic$$

proprietăți

$$evida(vid) = adevarat$$

$$evida(cons(X, L)) = fals$$

$$cap(cons(X, L)) = X$$

$$rest(cons(X, L)) = L$$

O listă cu n elemente se poate construi prin apeluri repetate ale constructorului *cons()*. Numărul de elemente ale unei liste se numește lungimea listei. Listele pot fi reprezentate folosind fie alocarea secvențială, fie alocarea înlănțuită.

Reprezentarea listelor folosind alocarea secvențială

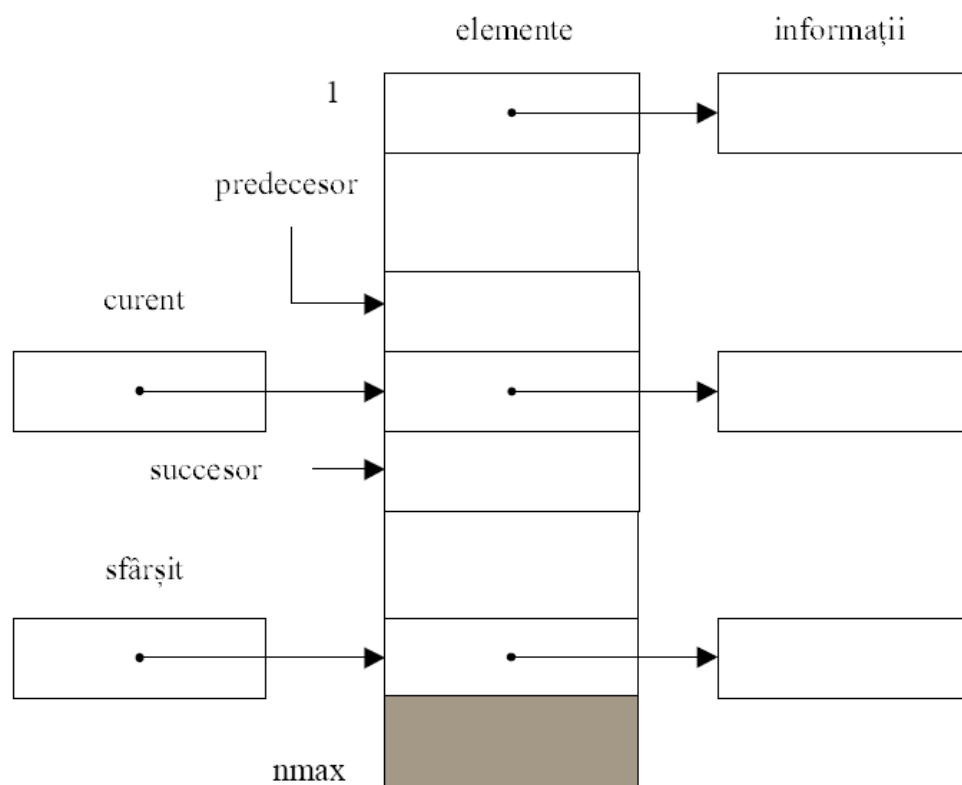
Cel mai simplu și mai natural mod de a reprezenta în memorie o listă liniară este de a plasa elementele listei în locații consecutive, un nod după altul, folosind în acest scop o structură de tip tablou. O componentă a tabloului poate stoca fie un nod al listei, fie un pointer la un nod al listei.

Accesul la un element din listă, parcurgerea listei sau adăugarea/ștergerea unui element la/de la sfârșitul listei se fac cu ușurință, în timp ce inserarea/ștergerea de elemente din interiorul listei sunt operații costisitoare, deoarece presupun deplasarea unor elemente.

Reprezentarea unei liste folosind alocarea secvențială folosește un tablou. Alegem în cele ce urmează soluția în care componentele acestui tablou sunt pointeri la informația utilă. Această metodă de reprezentare are avantajul că lista poate fi eterogenă, adică informațiile din noduri pot aparține unor tipuri diferite. Elementele listei vor ocupa pozițiile 1, 2, ..., din cadrul acestui tablou. În plus, trebuie cunoscută poziția ultimului element din listă.

Pentru parcurgerea listei este necesară utilizarea unei variabile numite cursor, care ne indică poziția elementului curent din listă (elementul care trebuie prelucrat). Dacă lista este folosită într-o aplicație în cadrul căreia o singură parcurgere a listei este activă la un moment dat, această variabilă poate fi adăugată la reprezentarea listei. În caz contrar, această variabilă va trebui reprezentată separat de listă. Vom presupune în continuare că ne aflăm în primul caz.

Pentru reprezentarea listei se folosește următoarea structură de date, care încapsulează variabila cursor, poziția ultimului element al listei și tabloul de pointeri în care se alocă elementele listei.

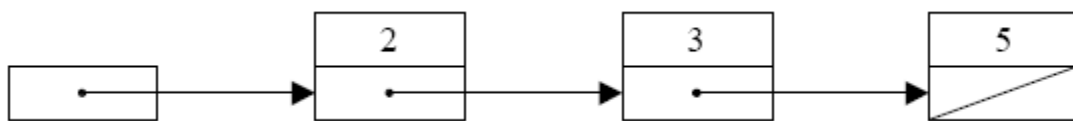


Reprezentarea listelor folosind alocarea secvențială

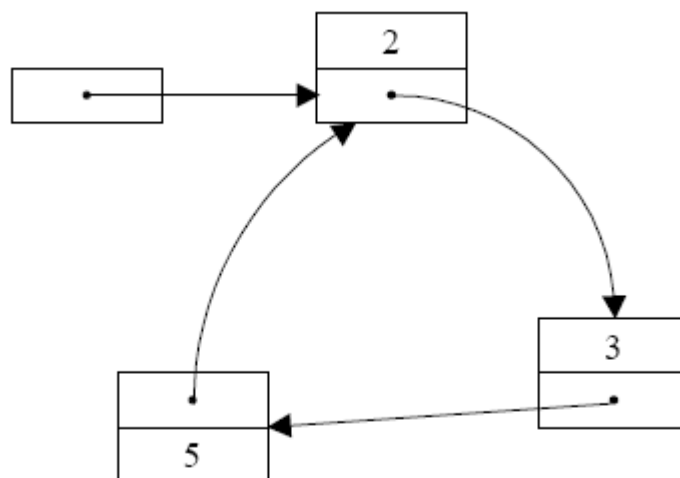
În cazul alocării înlănțuite, nodurile listei nu vor mai fi plasate în locații adiacente de memorie, ci în locații dispersate. Ordinea relativă a lor se păstrează prin includerea în cadrul fiecărui nod a unor informații de legătură. Spațiul suplimentar de memorie necesar includerii informațiilor de legătură reprezintă prețul plătit pentru înlăturarea neajunsurilor alocării secvențiale.

O listă liniară a cărei reprezentare internă folosește alocarea înlănțuită a nodurilor se numește listă înlănțuită sau listă legată.

Un nod al unei liste poate conține o singură legătură { către nodul succesori, caz în care vom spune că avem o listă simplu înlănțuită sau două legături { către nodurile predecesor și succesori, caz în care vom spune că avem o listă dublu înlănțuită. Se obișnuiește ca informația de legătură să fie indicată grafic printr-o săgeată orientată către informația "legată" de informația curentă. Spre exemplu, lista înlănțuită a primelor 3 numere prime este reprezentată grafic în figura următoare:

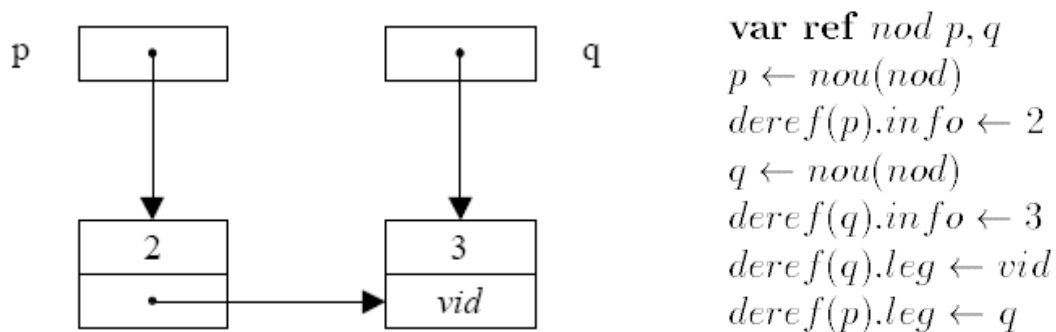


Pentru a se putea detecta sfârșitul listei, ultimul nod va conține o legătură nulă sau echivalent vidă (reprezentată printr-o celulă "taiată" în figură). O altă posibilitate este folosirea unei structuri în care ultimul nod să indice spre primul, ca în figura următoare. O listă înlănțuită reprezentată printr-o astfel de structură circulară se numește listă circulară.



Exemplu de listă circulară

Evident, putem avea liste circulare simple și liste circulare duble. Pentru reprezentarea informației de legătură se folosesc pointeri sau indici (numere întregi). În cazul în care informația de legătură se reprezintă prin pointeri, se obișnuiește ca nodurile listei să fie create dinamic. Legăturile se vor completa cu valorile pointerilor, obținute în urma operațiilor de alocare dinamică a memoriei.



```

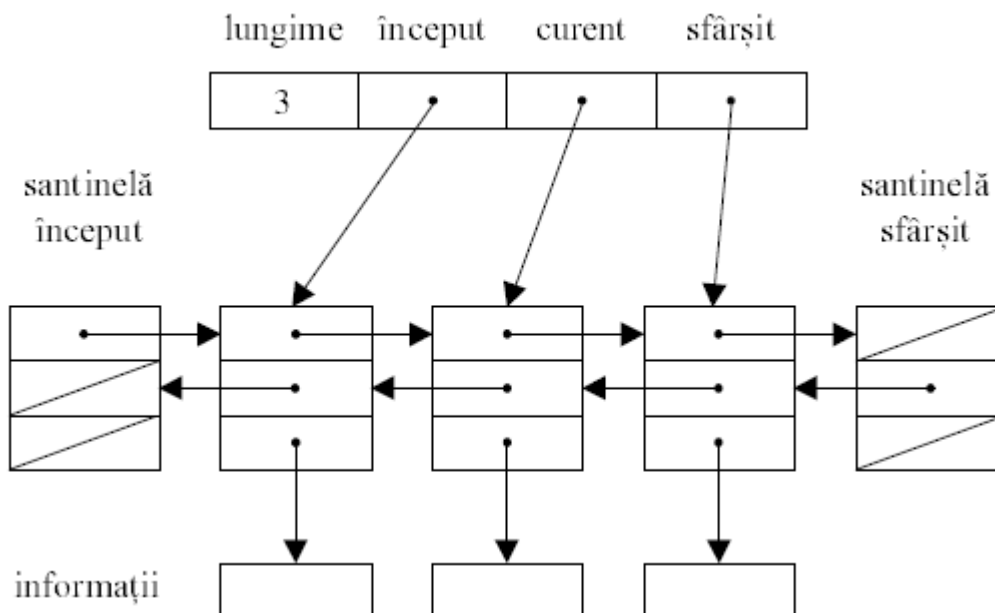
var ref nod p, q
p ← nou(nod)
deref(p).info ← 2
q ← nou(nod)
deref(q).info ← 3
deref(q).leg ← vid
deref(p).leg ← q

```

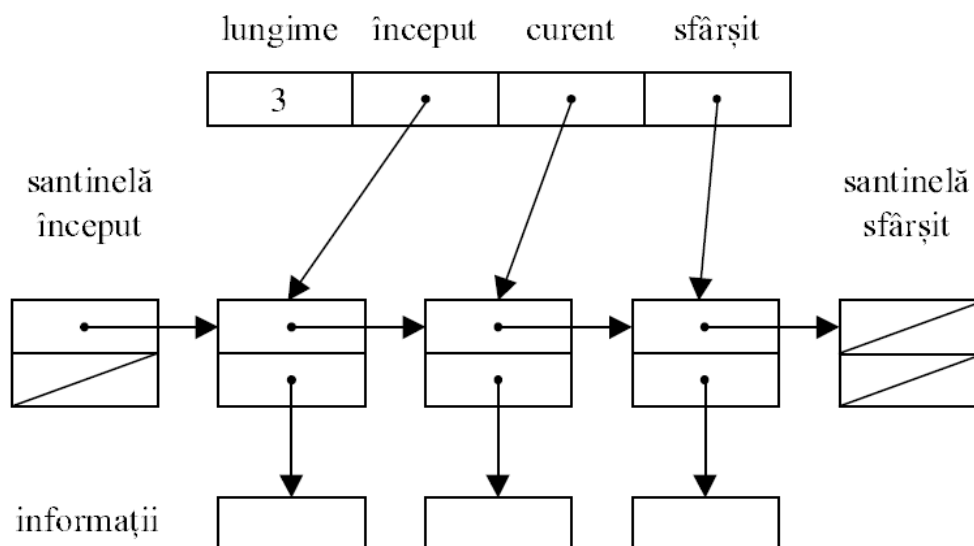
Exemplu de listă creată folosind alocare înlănțuită și programul aferent

Dacă se realizează implementarea listelor înlănțuite folosind pointeri și alocare dinamică, pentru reprezentarea unei liste vom folosi un obiect agregat care conține următoarele câmpuri: lungimea listei, pointerul la primul nod, pointerul la ultimul nod și pointerul la nodul curent (cursorul).

Pentru ca operațiile de inserare și ștergere să se facă la fel pentru orice nod al listei, se adaugă listei două noduri fictive numite și santinele, plasate la începutul și la sfârșitul listei (înaintea primului, respectiv după ultimul nod al listei). În acest fel, orice nod neredundant al listei va avea atât succesori cât și predecesori.



Reprezentarea listelor simplu înlănțuite cu pointeri și alocare dinamică



Reprezentarea listelor dublu înlănțuite cu pointeri și alocare dinamică

2.5. ARBORI. REPREZENTARE, PARCURGERE

Listele sunt potrivite pentru reprezentarea datelor organizate liniar. Dacă însă dorim să descriem date structurate ierarhic, simpla enumerare a obiectelor componente cu ajutorul unor liste este insuficientă.

Organizarea datelor sub formă ierarhică este frecvent întâlnită în cele mai diverse domenii aplicative. Câteva exemple sunt: organizarea administrativă sau managerială a unei societăți, planificarea meciurilor unei competiții sportive de tip turneu, structurarea unei cărți sau a directorului de fișiere dintr-un sistem de operare, reprezentarea expresiilor aritmetice și logice într-un compilator, în vederea evaluării lor. Generalizând, orice entitate poate fi descrisă la un nivel abstract printr-un obiect primitiv sau la un nivel detaliat sub forma unei ierarhii de obiecte componente.

Pentru definirea structurilor arborescente o atenție importantă trebuie acordată relației de "ramificare" sau echivalent "subordonare".

Se numește arbore o mulțime A de unul sau mai multe noduri astfel încât:

- i) există un nod special $\text{rad}(A) \in A$ numit rădăcina arborelui A
- ii) mulțimea celorlalte noduri din A cu excepția rădăcinii este partiționată în $n \geq 0$ mulțimi nevide și disjuncte A_i ; $1 \leq i \leq n$, care sunt la rândul lor arbori. $A_1 \dots A_n$ se numesc subarborii lui A . Dacă ordinea relativă a subarborilor $A_1 \dots A_n$ în punctul ii) al definiției este importantă, spunem că A este un arbore ordonat. În acest caz A_i este al i -lea subarbore al lui A pentru $1 \leq i \leq n$.

Dacă pentru a distinge între doi arbori se consideră că ordinea subarborilor este nerelevantă, atunci arborii se numesc orientați pentru a sugera faptul că are importanță doar orientarea arcelor (de la rădăcină spre rădăcinile subarborilor,) nu și ordinea relativă a acestora. Arborii orientați sunt un caz particular de grafuri orientate. Ei trebuie deosebiți de arborescențe, care sunt un caz particular de grafuri neorientate.

Definirea axiomatică a arborilor cu noduri de tipul N este următoarea:

structura $Arb(\mathcal{N})$

operații

$nod2arb : \mathcal{N} \rightarrow Arb(\mathcal{N})$

$ad_prim_fiu : Arb(\mathcal{N}) \times Arb(\mathcal{N}) \rightarrow Arb(\mathcal{N})$

$frate : Arb(\mathcal{N}) \rightarrow Arb(\mathcal{N})$

$prim_fiu : \{N \mid N \in Arb(\mathcal{N}), \neg efrunz\acute{a}(N)\} \rightarrow Arb(\mathcal{N})$

$grad : Arb(\mathcal{N}) \rightarrow Natural$

$rad : Arb(\mathcal{N}) \rightarrow Arb(\mathcal{N})$

$efrunza : Arb(\mathcal{N}) \rightarrow Logic$

proprietăți

$rad(nod2arb(R)) = R$

$grad(nod2arb(R)) = 0$

$prim_fiu(ad_prim_fiu(F, A)) = F$

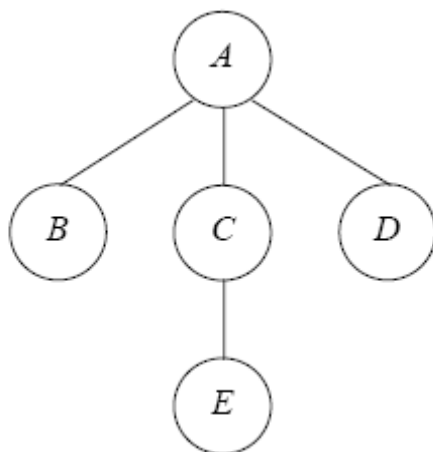
$grad(ad_prim_fiu(F, A)) = grad(A) + 1$

$rad(ad_prim_fiu(F, A)) = rad(A)$

$frate(prim_fiu(ad_prim_fiu(F, A))) = (efrunza(A) ? F : prim_fiu(A))$

$efrunza(A) = (grad(A) = 0)$

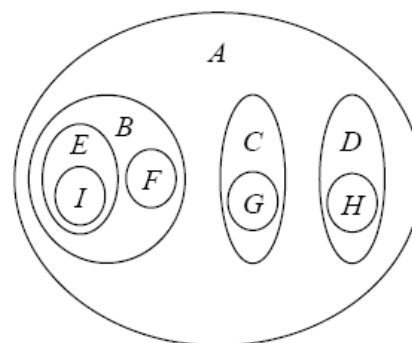
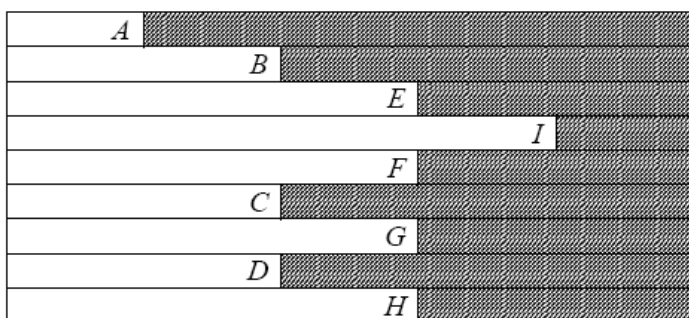
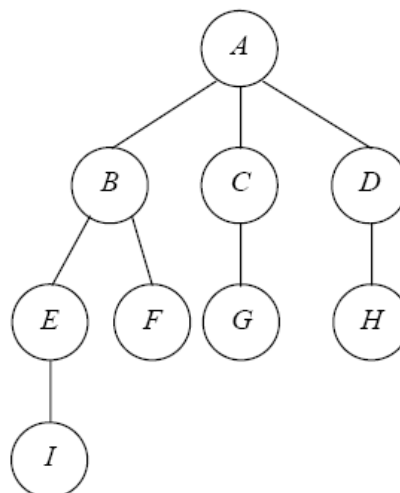
Un exemplu de creare a unui arbore folosind această modalitate de definire este:



```
ad_prim_fiu(  
  nod2arb(B),  
  ad_prim_fiu(  
    ad_prim_fiu(  
      nod2arb(E),  
      nod2arb(C)  
    ),  
    ad_prim_fiu(  
      nod2arb(D),  
      nod2arb(A)  
    ),  
  ),  
)
```

Structurile arborescente pot fi reprezentate grafic sau alfanumeric în diverse moduri care sugerează în fapt o aceeași structură.

<i>A</i>			
<i>B</i>		<i>C</i>	<i>D</i>
<i>E</i>	<i>F</i>	<i>G</i>	<i>H</i>
<i>I</i>			



$(A(B(E(I))(F))(C(G))(D(H)))$

Figura 5.5: Reprezentări posibile ale structurilor arborescente

Numărul de subarbori ai unui arbore se numește gradul arborelui.

Maximul gradelor subarborilor corespunzători fiecărui nod al unui arbore se numește aritatea arborelui. Nodurile cu gradul egal cu 0 se numesc noduri frunză sau noduri terminale. Celelalte noduri se numesc noduri neterminale.

Rădăcinile subarborilor unui arbore cu rădăcina X se numesc copiii sau fiii nodului X. Orice nod este tatăl sau părintele fiilor săi. Copiii unui aceluiași nod părinte se numesc frați.

Pentru orice nod X al unui arbore există o cale unică ce unește rădăcina R a arborelui cu nodul X. Toate nodurile de pe această cale, exceptându-l pe X, formează mulțimea strămoșilor lui X. Numărul strămoșilor lui X plus 1 reprezintă nivelul nodului X. Prin înălțimea sau adâncimea unui arbore vom înțelege nivelul maxim al nodurilor sale.

O mulțime de $n \geq 0$ arbori distincți se numește pădure. Dacă ordinea arborilor unei păduri este relevantă atunci avem o pădure ordonată.

Cel mai natural mod de a reprezenta o pădure este folosirea unei liste $Lista(Arb(N))$. Se observă că prin eliminarea rădăcinii unui arbore se obține o pădure formată din subarborii rădăcinii. Rezultă că o modalitate elegantă de a construi un arbore pornește de la rădăcina arborelui și pădurea subarborilor săi. Un astfel de constructor se definește axiomatic astfel:

$\text{cons_arb} : N \times \text{Lista}(\text{Arb}(N)) \rightarrow \text{Arb}(N)$

Această funcție are proprietățile următoare:

$\text{cons_arb}(R, \text{vid}) = \text{nod_2_arb}(R)$

$\text{cons_arb}(R, \text{cons}(N, L)) = \text{ad_prim_fiu}(N, \text{cons_arb}(R, L))$

Fiecare nod N al unei păduri se poate eticheta cu o secvență de numere $\alpha(N)$.

Mulțimea acestor secvențe luate împreună sugerează o structură arborescentă. Dacă pădurea are k arbori, rădăcinilor li se asociază numerele $1, 2, \dots, k$. Dacă α este secvența asociată unui nod oarecare de grad m , atunci copiilor li se vor asocia secvențele $\alpha.1, \alpha.2, \dots, \alpha.m$. Această metodă de codificare a nodurilor unei păduri se numește notația Dewey.

Spre exemplu, nodurile arborelui reprezentat axiomat anterior se codifică astfel:

$\alpha(A) = 1$

$\alpha(B) = 1.1$

$\alpha(C) = 1.2$

$\alpha(D) = 1.3$

$\alpha(E) = 1.2.1$

Un caz special de arbore frecvent întâlnit în diverse domenii aplicative este arborele binar. Într-un astfel de arbore fiecare nod are maxim doi subarbori; mai mult, când este prezent un singur subarbore, se face distincție între subarborii stâng și subarborii drept. Se numește arbore binar o mulțime finită de noduri care fie este vidă, fie constă dintr-o rădăcină și elementele a doi arbori binari disjuncți numiți subarborii stâng și subarborii drept ai rădăcinii.

Prelucrarea informațiilor dintr-un arbore implică parcurgerea arborelui. Se numește parcurgere o metodă de examinare sistematică a nodurilor unui arbore astfel încât fiecare nod să fie vizitat exact o singură dată. Parcurgerea arborilor ne oferă o aranjare liniară a nodurilor, astfel încât în orice moment vom ști precis care este următorul nod care va fi vizitat și prelucrat.

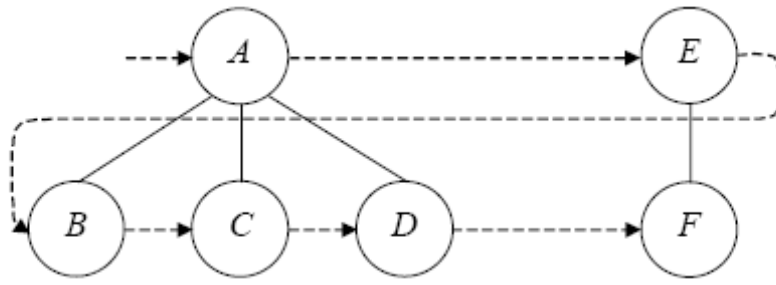
În cazul parcurgerii în lărgime, se prelucrează mai întâi informația din rădăcini, în ordinea în care arborii apar în cadrul pădurii, după care sunt prelucrate, de la stânga la dreapta, nodurile aflate pe primul nivel în arborii pădurii, apoi cele aflate pe al doilea nivel, ș.a.m.d. Parcurgerea în lărgime presupune folosirea unei cozi. Inițial coada este vidă. Dacă atât coada cât și pădurea sunt vide nu se efectuează nici o prelucrare. Dacă pădurea este nevidă, se depune primul arbore al pădurii în coadă și parcurgerea continuă cu restul pădurii și noua coadă. Altfel (pădurea este vidă), dacă coada este nevidă, se extrage un arbore din coadă, se vizitează rădăcina și se continuă parcurgerea cu pădurea subarborilor arborelui extras din coadă și cu restul cozii.

În cazul parcurgerii în adâncime, copiii unui nod sunt vizitați tot de la stânga la dreapta, însă trecerea de la nodul curent la fratele din dreapta are loc numai după ce a fost parcurs tot subarborii cu rădăcina în nodul curent. Pentru a memora informațiile relative la punctul de revenire (nodul tată și următorul fiu neprelucrat al său) se folosește o stivă. Operația de parcurgere în adâncime decurge absolut la fel ca și cea de parcurgere în lărgime, cu deosebirea că în locul operațiilor cu cozi se folosesc operații cu stive. În plus, la parcurgerea în preordine rădăcina este prelucrată înaintea parcurgerii subarborilor săi, iar la parcurgerea în postordine rădăcina este prelucrată după parcurgerea subarborilor săi.

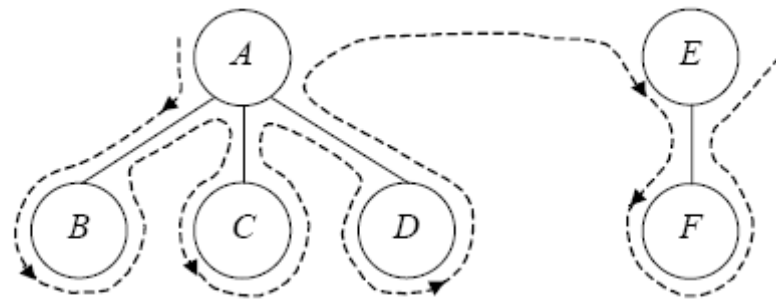
În cazul arborilor binari este consacrată parcurgerea în adâncime. Datorită faptului că un arbore binar este compus din 3 componente: rădăcina R , subarborii stâng S și subarborii drept D , rezultă că se pot defini $3! = 6$ parcurgeri în adâncime, simbolizate sugestiv prin următoarele mnemonice: $SRD, RSD, SDR, DSR, DRS, RDS$. Dintre acestea, doar primele 3 au denumiri consacrate și anume: inordine, preordine și respectiv postordine. Prefixele in, pre și post sugerează ordinea relativă de vizitare a rădăcinii în raport cu cei doi subarbori.

Arborii binari se reprezintă fie utilizând alocarea secvențială, fie utilizând alocarea înlănțuită.

În continuare sunt prezentate grafic câteva exemple de parcurgere a arborilor.



Parcurgerea în lărgime a unei păduri



Parcurgerea în adâncime(preordine) a unei păduri

CAP.3. SISTEME DE OPERARE

3.1. NOȚIUNI INTRODUCATIVE PRIVIND SISTEMELE DE OPERARE

3.1.1 Definiție și rol

Se poate defini sistemul de operare, ca fiind un *set de programe specializate* ce asigură legătura funcțională între elementele componente ale unui sistem de calcul, controlând și coordonând utilizarea resurselor acestuia între diferitele programe de aplicații ale utilizatorilor. El acționează ca o interfață între utilizator și hardware-ul sistemului de calcul, mascând complexitatea acestuia.

Rolul său este, pe de o parte, de a crea un mediu în care utilizatorul să poată executa programe cu mai multă ușurință, iar pe de altă parte, de a asigura utilizarea eficientă a hardware-ului. Aceste două obiective sunt de multe ori contradictorii.

Sistemul de operare furnizează instrumentele cu ajutorul cărora să fie folosite în mod corespunzător resursele de bază ale sistemului de calcul (hardware, software, date). Din acest punct de vedere sistemul de operare poate fi privit ca *administrator al resurselor* pe care le alocă în funcție de necesități programelor și utilizatorilor. Deoarece pot exista multe cereri (uneori conflictuale) adresate resurselor, sistemul de operare are menirea de a decide strategia de alocare, astfel încât exploatarea sistemului de calcul să fie corectă și eficientă.

O altă definiție a sistemului de operare, ușor diferită de cea anterioară, pune accentul pe necesitatea *controlului asupra dispozitivelor de intrare/ieșire și a programelor utilizatorilor*. În această abordare, un sistem de operare este un *program de control* care urmărește executarea programelor utilizatorilor pentru a putea preveni eventualele erori, precum și folosirea necorespunzătoare a sistemului de calcul și are deci ca principală sarcină operarea și controlul cu și asupra dispozitivelor de I/E.

Sistemul de operare poate fi gândit ca o *mașină virtuală* interpusă între programele utilizator și hardware furnizând astfel o mașină extinsă mult mai ușor de programat decât hardware-ul brut.

De asemenea, sistemul de operare poate fi gândit ca un *sistem care reacționează, care răspunde*, la cereri ce sosesc de la programele utilizator (apeluri supervisor + excepții) și de la dispozitivele periferice (întreruperi).

3.1.2. Funcții

Un prim *set de funcții* pe care trebuie să le îndeplinească un sistem de operare, sunt legate de rolul său de *interfață între hardware și utilizatori*. Astfel, sistemul de operare trebuie să fie capabil să asigure:

1) *posibilitatea de pregătire și lansare în execuție a programelor de aplicație*; În acest scop, un sistem de operare trebuie să dispună de cel puțin următoarele componente:

- un *editor de texte* pentru a introduce și modifica un *program sursă* (PS) scris într-un limbaj de programare;
- un *translator* pentru limbajul de programare folosit (asamblor, compilator sau interpretor), care să traducă instrucțiunile programului sursă, într-o formă recunoscută de sistemul de calcul (forma binară) - *program obiect* sau *module obiect* (PO);
- un *editor de legături* care să realizeze legătura dintre diverse module obiect, sau să apeleze la module obiect din bibliotecile sistemului, respectiv la modulele obiect din biblioteca utilizatorului - care au fost catalogate în prealabil, pentru a construi structură pe segmente impusă de sistemul de calcul în vederea execuției programelor (*program obiect executabil-POE*).

Odată construită structura pe segmente, programul va fi gata de execuție, ceea ce va implica încărcarea acestuia în memoria internă și execuția efectivă; componenta sistemului de operare ce realizează acest lucru se numește *încărcător*.

2) *alocarea resurselor necesare execuției programelor* prin:

- identificarea programelor ce se execută și a necesarului de resurse;
- alocarea memoriei interne și a dispozitivelor periferice;
- identificarea și protecția colecțiilor de date.

3) *acordarea unor facilități prin programe utilitare de interes general*:

- gestiune cataloage (directoare, subdirectoare) și fișiere;
- creare, modificare, copiere, mutare, ștergere, recuperare directoare și fișiere;
- sortare/interclasare;
- depistare și eliminare viruși informatici și altele.

4) *planificarea execuției mai multor programe* (multiprogramare) după anumite criterii, în vederea utilizării eficiente a unității centrale (UC);

5) *coordonarea execuției mai multor programe ce se execută simultan*, prin urmărirea modului de execuție a instrucțiunilor programelor, depistarea și tratarea erorilor, lansarea în execuție a operațiilor de intrare/ieșire;

6) *asistarea execuției programelor de către utilizator*, prin comunicația sistem de calcul-utilizator atât la nivel hardware cât și la nivel software;

7) *asigurarea organizării și protecției datelor în memorie*;

8) *posibilitatea generării unui sistem de operare* pe măsura configurației existente.

Un al doilea *set de funcții* pe care trebuie să le îndeplinească un sistem de operare, sunt legate de *utilizarea eficientă a resurselor sistemului de calcul*, lucru ce impune o gestionare corespunzătoare a acestora. Ca urmare, sistemul de operare trebuie să asigure:

1) *gestiunea memoriei*, care constă în:

- evidența acestei resurse - câtă memorie este alocată și pentru care programe;
- cărui proces i se alocă memorie, la ce moment și în ce cantitate - în cazul multiprogramării;
- alocarea de părți din memorie și asigurarea metode de acces și protecție pentru procesele solicitante;
- dezalocarea zonele de memorie alocate.

2) *gestiunea procesorului*, se referă la:

- evidența procesoarelor și stărilor acestora;
- decide cine poate să utilizeze procesorul, la ce moment de timp și pentru cât timp;
- alocă procesorul la un proces prin pregătirea și încărcarea unor registre hardware;
- retrage alocarea când procesul renunță la utilizarea procesorului, s-a terminat sau a depășit cuanta de timp alocată.

3) *gestiunea dispozitivelor periferice*, care constă în următoarele activități:

- evidența dispozitivelor, a unităților de control și a canalelor de I/E;
- decide metoda cea mai eficientă de alocare a dispozitivelor periferice;
- dacă are loc o utilizare simultană, decide cine folosește resursa și cât timp;
- alocarea dispozitivelor periferice și inițiază operația de intrare/ieșire;
- dezalocarea dispozitivelor periferice la terminarea execuției operațiilor de intrare/ieșire.

4) *gestiunea informației*, care se materializează în:

- evidențierea resursei (informația), localizarea ei, utilizarea, starea, etc.;
- decide cine utilizează informația, impune protecția cerută și oferă rutine de acces necesare;
- alocă resursele prin deschiderea fișierului;
- dezalocă resursele prin închiderea fișierului.

3.1.3. Componentele sistemelor de operare

Din punctul de vedere al interacțiunii cu componentele hardware ale sistemului de calcul și după modul de implementare a software-ului, sistemul de operare este organizat pe două niveluri: *fizic* și *logic*.

Nivelul fizic oferă servicii privind lucrul cu componentele hardware ale sistemului de

calcul și cuprinde acele elemente ce depind de configurația sistemului.

Nivelul logic include partea de programe prin care utilizatorului poate exploata sistemul de calcul. Comunicarea utilizatorului cu sistemul de calcul se realizează prin comenzi adresate sistemului de operare sau prin intermediul instrucțiunilor programelor pe care le execută; invers, comunicarea se realizează prin intermediul mesajelor transmise de sistemul de operare către utilizator.

Din punct de vedere funcțional, programele sistemului de operare se împart în două categorii:

- *programe de comandă și control* (cunoscute și sub numele de *monitoare, supervizoare* sau *executive*) - care vizează optimizarea utilizării resurselor sistemului de calcul prin coordonarea și controlul tuturor funcțiilor acestuia;
- *programe de serviciu* - care sunt destinate optimizării accesului la resursele sistemului de calcul, fiind utilizate de programator pentru dezvoltarea programelor de aplicații.

Programele de comandă și control, coordonează activitatea celorlalte componente ale sistemului de operare, îndeplinind următoarele funcții majore:

- *gestiunea resurselor fizice și logice* (gestiunea memoriei, gestiunea fișierelor, gestiunea fizică a intrărilor și ieșirilor, gestiunea proceselor, evidența gradului de utilizare a componentelor, etc.);
- *planificarea, lansarea și urmărirea execuției* (planificarea lucrărilor și alocarea resurselor, evidența lucrărilor executate, a utilizatorilor și a resurselor consumate, etc.);
- *depistarea și tratarea evenimentelor la execuție* (evidența erorilor hardware, evidența întreruperilor etc.).

Cele mai frecvent utilizate componente ale supervisorului sunt încărcate în memoria internă încă de la generarea sistemului de operare, fiind păstrate în aceasta pe tot parcursul execuției aplicațiilor de către sistemul de calcul; aceste programe se numesc *rutine rezidente*, și formează *nucleul* sistemului de operare; celelalte componente rămân în memoria externă fiind apelate și executate numai atunci când sunt solicitate de către nucleu (se numesc *rutine tranziente*) asemenea oricărui program de aplicație (figura 3.1):

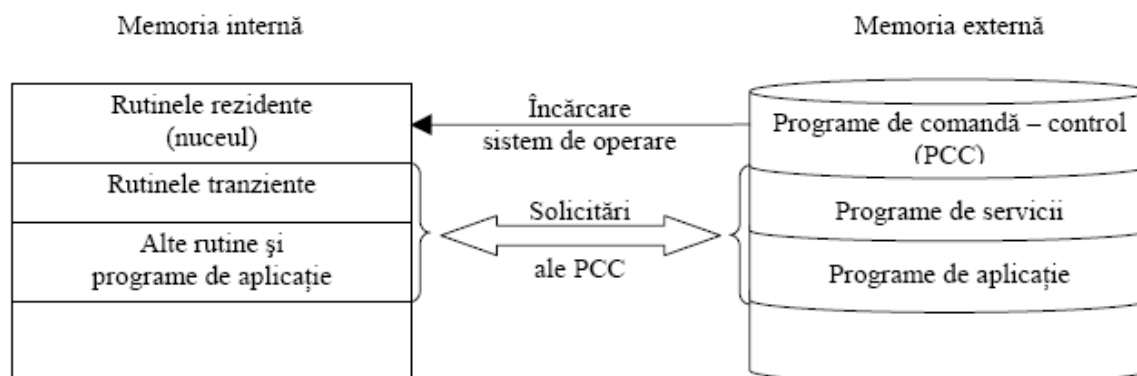


Fig. 3.1 Rutine rezidente și rutine tranziente

Programele de serviciu se execută sub supravegherea programelor de comandă și control și pot fi diferite de la un sistem de operare la altul, sau chiar între versiunile aceluiași sistem de operare. Ele permit utilizatorului să folosească resursele fizice și logice ale sistemului de calcul pentru realizarea aplicațiilor. Cele mai utilizate programe din această categorie sunt:

- *Translatorii de limbaje* - au rolul de a transforma instrucțiunile programelor sursă (scrise de utilizatori într-un limbaj de programare), în coduri executabile de calculator (format obiect). Din această categorie fac parte:
 - *asamblearele*, care au rolul de a traduce programele sursă, scrise în limbaje de asamblare, în programe obiect executabile. Aceste programe

- sunt specifice unui anumit tip de sistem de calcul fiind determinate de limbajul mașină al acestuia.
- *compilatoarele*, care sunt specifice limbajelor de programare de nivel înalt, independent de sistemul de calcul, asigurând traducerea programelor sursă în programe obiect.
 - *interpretoarele* care analizează și execută pas cu pas instrucțiunile programului sursă, permițând o punere mai rapidă la punct a programelor.
- *Editorul de legături* - are rolul de a transforma modulele obiect (obținute cu ajutorul translaatoarelor) în programe executabile.
 - *Bibliotecarul* - asigură crearea, gestionarea și întreținerea bibliotecii sistem (care conține programele sistemului de operare) și a bibliotecilor utilizator. Bibliotecile de programe sunt colecții de programe organizate sub forma unor fișiere partajate în scopul utilizării lor ulterioare;
 - *Programele de încărcare* - sunt componente ale programelor de serviciu, cu rolul de a încărca în memoria internă (RAM), în vederea execuției, programele obiect executabile;
 - *Generatorul sistemului de operare* - permite utilizatorului să genereze un sistem de operare compatibil cu configurația hardware de care dispune (memorie internă, echipamente periferice utilizate, tipuri de interfețe etc.) și cu modalitățile de exploatare adoptate în funcție de opțiunile domeniului de utilizare;
 - *Depanatorul* este un program de serviciu ce oferă utilizatorului mijloace convenabile pentru depanarea și controlul operațiilor programului.
 - *Programele de organizare a colecțiilor de date* - asigură operațiile de intrare /ieșire prin colecții de date (fișiere, baze de date, depozite de date);
 - *Mediile de programare* - sunt programe destinate automatizării procesului de construire și testare a programelor. Cu ajutorul mediilor de programare se realizează: editarea, compilarea și, eventual, editarea de legături, lansarea în execuție și depanarea unui program;
 - *Programele utilitare* - s-au dezvoltat și diversificat odată cu perfecționarea calculatoarelor, a modalităților de exploatare și a domeniilor de aplicare, fiind adesea incluse în sistemul de operare și destinate să extindă funcționalitatea acestuia.

3.1.4. Moduri de operare

Pentru a proteja sistemul de calcul împotriva unei utilizări necorespunzătoare, folosirea de către programele utilizatorilor a anumitor instrucțiuni mașină este restricționată, acestea putând fi efectuate numai de către sistemul de operare. Astfel, programele utilizator nu pot:

- să adreseze dispozitivele de I/E, direct;
- să utilizeze instrucțiunile care manipulează starea memoriei;
- să seteze bitul de mod care determină modul utilizator sau supervisor;
- să oprească calculatorul (instrucțiunea Halt).

Instrucțiunile mașină respective se numesc *instrucțiuni privilegiate*.

Pentru ca sistemul de calcul să se comporte diferit față de sistemul de operare pe de o parte, și față de programele utilizatorilor, pe de altă parte, au fost adoptate două moduri de operare distincte, și anume: *modul utilizator*, în cadrul căruia nu sunt utilizate instrucțiuni privilegiate și *modul supervisor* (numit și *mod monitor* sau *mod sistem*) în cadrul căruia toate instrucțiunile pot fi executate. Această distincție între modurile de operare este o consecință a necesității realizării unei protecții față de utilizarea necontrolată a instrucțiunilor, și ca urmare, arhitectura oricărui sistem de operare trebuie să suporte cel puțin cele două moduri: *utilizator* și cel *supervisor*.

Deoarece nucleul reprezintă partea protejată a sistemului de operare el rulează în modul

supervizor, protejând astfel structurile de date critice ale sistemului de operare și regiștrii de dispozitiv împotriva programelor utilizator.

Pentru identificarea modului curent de operare se utilizează un *bit de mod* din cadrul unui registru al unității centrale, registru protejat. Semnificația bitului de mod respectiv este: 0 = supervizor, 1 = utilizator.

Transferul din modul utilizator în modul supervizor poate să se întâmple din cauza execuției explicite a unei *instrucțiuni apel supervizor* (sau sistem) (numită și SVC sau TRAP) sau din cauza unei întreruperi hardware sau a unei excepții. Transferul înapoi din modul supervizor în modul utilizator apare din cauza unei instrucțiuni return explicite (uzual numită REI, Return din întrerupere).

3.1.5. Întreruperi

Necesitatea *sistemului de întreruperi* devine evidentă în momentul în care studiem modul în care se execută programele. Uzual, procesorul execută instrucțiunile într-o ordine dată de următoarele reguli:

1. dacă instrucțiunea curentă este una de salt, va fi executată în continuare instrucțiunea de la adresa la care se face saltul;
2. în caz contrar, va fi executată în continuare instrucțiunea aflată în memorie la adresa imediat următoare după instrucțiunea curentă.

Ca o concluzie generală, întotdeauna o instrucțiune care face parte dintr-un program va fi urmată de o altă instrucțiune din același program. Până aici nu există nici o posibilitate de a părăsi programul aflat în execuție decât dacă acesta se termină singur. Acest model corespunde în general cerințelor, deoarece un program aflat în execuție rulează în majoritatea timpului fără a ține cont de existența sistemului de operare, totuși, acesta trebuie să poată interveni în anumite situații bine definite, cum ar fi:

- o cerere explicită adresată de programul de aplicație, privind efectuarea unui anumit serviciu de către sistemul de operare, serviciu pe care aplicația nu-l poate efectua singură;
- o cerere de întrerupere venită din partea unui dispozitiv periferic, care poate să nu aibă legătură cu programul aflat în execuție, dar care trebuie tratată imediat (altfel datele se pot pierde);
- o operație executată de UC care s-a terminat anormal (de exemplu o operație de împărțire la zero, depășirea de capacitate, un acces anormal la o zonă de memorie, etc.), ceea ce indică încercarea unui program de a efectua o acțiune nepermisă.

Sistemul de operare va lăsa deci orice program să se execute fără interferențe până la apariția uneia din situațiile descrise mai sus, dar în acest moment trebuie să preia imediat controlul. Soluția este, așa cum am precizat deja, de natură hardware și este reprezentată de *sistemul de întreruperi*. Concret, acesta oferă posibilitatea întreruperii execuției programului curent, în anumite condiții, pentru a trata o sarcină considerată ca fiind mult mai urgentă.

Fiecăreia din situațiile prezentate mai sus îi corespunde unul tipurile de întrerupere:

- *întreruperi software* (externe);
- *întreruperi hardware* (externe);
- *excepții* (întreruperi hardware interne).

Deoarece anumite întreruperi pot fi mai importante de cât altele ele dovedindu-se deci a fi prioritare, și ca urmare este necesară stabilirea unei anumite ierarhii de priorități a acestora.

Unitățile centrale ale sistemelor de calcul sunt prevăzute cu instrucțiuni ce autorizează sau interzic întreruperile în anumite cazuri. Astfel, dacă programul aflat în execuție nu trebuie să fie întrerupt, luarea în considerare și tratarea întreruperilor va fi interzisă pentru a preveni perturbarea acestuia. Totuși anumite întreruperi nu vor fi interzise, fie datorită necesității lor, fie datorită nivelului lor de prioritate. Exemplul cel mai reprezentativ este cazul întreruperii datorate

întreruperii tensiunii de alimentare, care nu va fi interzisă. Asemenea întreruperi sunt numite *nemascabile*.

Din contră, o întrerupere se spune că este *mascabilă* când există posibilitatea ca unitatea centrală să o ignore. Putem astfel să mascăm, la un moment dat, anumite întreruperi pentru conservarea derulării programului în curs față de orice întrerupere intempestivă (cu excepția evidentă a întreruperilor nemascabile).

Tratarea unei întreruperi, indiferent care este cauza care a produs-o, se derulează în general de următoarea manieră:

- se recepționează de către unitatea centrală o cerere de întrerupere internă sau externă;
- după acceptarea ei are loc sfârșitul tratării instrucțiunii în curs de execuție;
- se salvează starea sistemului, adică se salvează conținutul diverselor regiștri (numărătorul de program, etc.), în așa fel încât să se poată relua execuția programului întrerupt din starea în care se găsea la momentul întreruperii;
- se încarcă numărătorul de program cu valoarea adresei primei instrucțiuni a sub-programului (rutinei) de tratare asociat acestei întreruperi.
- sub-programul de tratare a întreruperii odată terminat provoacă restaurarea stării în care găsea sistemul la momentul luării în considerare a întreruperii.

Evident, rutinele care tratează situațiile generatoare de întreruperi fac parte din sistemul de operare, care poate astfel rezolva problemele apărute.

Apeluri sistem (supervizor)

Una din sursele întreruperilor, prezentate anterior, o constituie solicitările formulate în mod explicit de programele de aplicații către sistemul de operare, pentru efectuarea anumitor servicii. De ce este însă necesar ca aceste servicii să fie implementate de către sistemul de operare și nu pot fi lăsate în seama programelor? În primul rând, unele operații uzuale (afișarea, căutarea pe disc etc.) se desfășoară întotdeauna în același mod; deci, în loc de a scrie practic aceeași rutină în fiecare program, este mai economic de a o scrie o singură dată ca parte a sistemului de operare, astfel ca toate aplicațiile să o poată utiliza. De altfel, apelul către un asemenea serviciu oferit de sistem nu se deosebește prea mult de apelul către o procedură sau funcție din același program.

Pe de altă parte, o serie de acțiuni, în special accesul la dispozitivele periferice, prezintă riscuri considerabile pentru întregul sistem de calcul în cazul în care nu sunt realizate corect. Nu este deci convenabil de a permite programelor de aplicații să realizeze singure acțiunile din această categorie; se preferă ca activitățile de acest tip să fie îndeplinite numai prin intermediul unor rutine incluse în sistemul de operare. Pentru a pune în practică o asemenea abordare, trebuie să se poată interzice pur și simplu realizarea anumitor operații de către programele de aplicații. Din nou este necesar un suport hardware.

În acest moment putem studia ce se întâmplă atunci când un program cere sistemului de operare furnizarea unui anumit serviciu. O asemenea cerere poartă numele de *apel sistem* (system call) și ea reprezintă interfața dintre un program aflat în execuție și sistemul de operare. Apelurile sistem sunt disponibile atât sub forma de instrucțiuni scrise în limbaj de asamblare, cât și ca funcții sau subrutine apelabile în cadrul programelor scrise în limbaj de nivel înalt. Apelurile sistem pot fi generate prin intermediul unei rutine specializate, sau direct, și ele sunt cerute în mod explicit de către un program utilizator pentru a trece din modul utilizator în modul supervizor.

Apelurile sistem creează, distrug și utilizează diferite obiecte logice gestionate de către sistemul de operare. Procesele și fișierele sunt cele mai importante obiecte, dar există apeluri supervizor și pentru administrarea memoriei, efectuarea de intrări/ieșiri pe un terminal sau o imprimantă, etc.

Apelul sistem este tratat de către hardware ca o *întrerupere software*, controlul fiind dat unei rutine a monitorului rezident, iar bitul de mod primește valoarea corespunzătoare modului

supervizor.

Fiecărui apel sistem îi corespunde o procedură de bibliotecă pe care programul utilizatorului poate să o apeleze. Această procedură plasează parametri de apel supervizor într-un loc predefinit cum ar fi registrele unității centrale apoi execută o instrucțiune TRAP (un apel de procedură protejat într-un anumit fel) pentru a activa sistemul de operare. Procedura de bibliotecă are ca scop de a masca detaliile instrucțiunii TRAP și de a face să apară apelurile supervizor ca și apeluri de proceduri obișnuite. Atunci când sistemul de operare preia controlul după TRAP, el verifică validitatea parametrilor și efectuează în acest caz tratarea cerută. Când a terminat, sistemul de operare plasează un cod de stare într-un registru indicând dacă tratarea a reușit sau a eșuat, apoi execută instrucțiunea RET (REturn fromTrap) pentru a reda controlul procedurii din bibliotecă. Această procedură redă controlul apelantului (programului utilizator) și retrimite codul de stare ca o valoare de retur a funcției. Anumite valori complementare sunt de asemenea returnate în parametrii de apel.

Numărul și tipul apelurilor sistem variază de la un sistem de operare la altul.

◇ **Excepții**

Multe dintre erorile de programare pot fi detectate de către hardware și sunt controlate, în mod normal, de către monitorul rezident. Dacă într-un program apare o instrucțiune nepermisă sau o greșeală de adresare a memoriei, hardware-ul va genera o întrerupere către monitorul rezident, numită *excepție* ce va determina comutarea din modul utilizator în modul supervizor. Așa cum s-a arătat, termenul de *excepție* este folosit pentru a desemna o *întrerupere hardware internă*, generată de obicei ca urmare a apariției unei erori de program cum ar fi: încercare efectuării unei împărțiri cu zero, utilizarea unei adrese de memorie ilegale, scrierea într-o locație de tip read-only, etc. Acționând ca o întrerupere, aceasta va determina memorarea codului program asociat ultimei instrucțiuni executate și va da controlul unei rutine de tratare a excepției din monitorului rezident. Înainte de a aborda următorul program, monitorul va executa în mod automat un vidaj de memorie și de registre, care poate fi utilizat pentru depanarea programului abandonat.

3.2. PROCESE CONCURENTE

3.2.1. Noțiunea de proces

În cadrul sistemelor de operare moderne, se consideră ca unitate de lucru, *procesul secvențial*. Acesta, definit ca entitate activă, este reprezentat de un program aflat în execuție împreună cu datele sale și cu toate celelalte informații necesare execuției, instrucțiunile fiind parcurse una câte una, la momente de timp diferite.

Trebuie făcută deosebirea dintre proces și program. *Procesul are un caracter dinamic*, el precizează o secvență de activități¹ în curs de execuție, iar *programul are un caracter static*, el numai descrie textual această secvență de activități.

Procesele concurente numite și *procese paralele*, presupun că la un moment dat, mai multe programe pot fi urmărite între punctul de începere și terminare a execuției; în acest caz procesele interacționează în două moduri:

- *indirect*, prin concurența la aceeași resursă a sistemului;
- *direct*, prin utilizarea simultană a acelorași resurse.

¹*Activitatea* (task) este o unitate de lucru internă creată de sistemul de operare, atunci când o etapă din lucrare este acceptată de sistemul de calcul. *Lucrarea* reprezintă o colecție de activități ce se execută de către sistemul de calcul.

3.2.2 Stările unui proces

Pe durata execuției lor procesele trec printr-o serie de stări discrete, o stare fiind definită ca o activitate curentă a procesului. Aceste stări sunt:

- **starea de execuție** (*running*) - un proces în starea de *execuție* are alocate toate resursele necesare rulării, inclusiv UC-ul. În cazul unui sistem cu un singur procesor, vom avea un singur proces activ la un moment dat în sistem. Procesul curent activ execută secvența sa de instrucțiuni mașină, putând cere sistemului de operare servicii ca operații I/E sau sincronizare printr-un semnal. Depinde de politica de planificare dacă sistemul returnează controlul acestui proces după terminarea serviciului sau alege un altul (din procesele în starea pregătit) pentru execuție;
- **starea gata de execuție** (*ready*) - un proces este în starea *pregătit (gata de execuție)* când are alocate toate resursele de care are nevoie, cu excepția UC-ului. De obicei, în această stare ajung procesele imediat după creare. Toate procesele în starea gata de execuție așteaptă ca sistemul de operare să le aloce UC-ul. Un modul al sistemului de operare, numit *planificator*, alege unul din aceste procese ori de câte ori UC-ul devine liber pentru a rula un nou proces.
- **starea blocată** (*blocked*) - procesul se află în așteptarea realizării unui eveniment (de exemplu, terminarea unei operații de I/E).

Schimbarea stării unui proces poate fi determinată fie de procesul însuși (de exemplu prin efectuarea unui apel sistem), fie datorită apariției unui eveniment extern (de exemplu când unitatea de comandă generează o întrerupere de ceas).

Să considerăm un sistem cu o singură unitate de centrală. La un moment dat aceasta poate executa un singur proces, dar mai multe procese pot fi gata de execuție iar altele pot fi blocate. Din acest motiv se stabilește o listă a proceselor gata de execuție și o listă a proceselor blocate. Lista proceselor gata de execuție este ținută în ordinea priorităților astfel încât primul proces ce va intra în prelucrarea unității de comandă va fi primul din listă. Lista proceselor blocate este neordonată deoarece aceste procese nu se vor debloca în ordinea din listă ci în ordinea în care se vor finaliza evenimentele ce au determinat blocarea lor. Există situații în care mai multe procese pot fi blocate așteptând realizarea aceluiași eveniment, în acest caz fiind necesară introducerea unor priorități.

Când un proces este creat el este trecut la sfârșitul listei proceselor gata de execuție. Pe măsură ce procesele intră în execuție, lista lor se decalează către început. Al doilea proces va lua locul primului, al treilea va lua locul celui de-al doilea, ș.a.m.d. Când procesul trece în capul listei și când unitatea de comandă a devenit disponibilă pentru a-l prelua, se spune despre proces că se găsește în starea de tranziție de la starea gata de execuție la starea de execuție. După cum am mai spus, un proces este format din cod și date, fiind caracterizat de attribute și stare dinamică. Attributele asociate unui proces pot fi asigurate de către programator sau de către sistemul de operare și includ prioritatea și drepturile de acces. Prezentăm mai jos diagrama de tranziție a stărilor proceselor (figura 3.2):

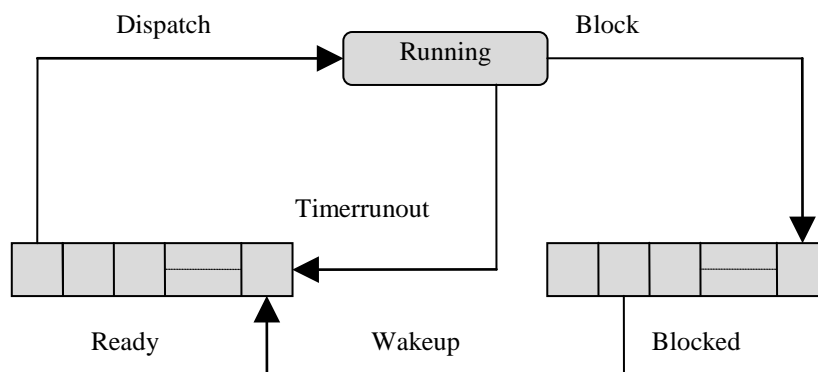


Fig 3.2. Stările de tranziție ale unui proces

Operația de asignare (atribuire) a unității de centrale către primul proces gata din lista proceselor gata de execuție se numește *expediere (dispatch)* și este executată de o entitate a sistemului de operare numită *expeditor (dispatcher)*. Această stare de tranziție se indică astfel:

dispatch (nume_proces): ready → *running*

Pentru a preveni situația în care un singur proces să monopolizeze sistemul de calcul, sistemul de operare este prevăzut cu o componentă software numită *ceas de control* care permite execuția procesului un interval de timp bine precizat. Dacă procesul nu eliberează unitatea de comandă înaintea expirării timpului, ceasul de control generează o întrerupere care permite unității de centrale să treacă procesul din starea de execuție în starea gata de execuție și să lanseze în execuție primul proces din lista celor gata de execuție.

Această stare de tranziție se indică astfel:

timerrunout (nume_proces_1): running → *ready*

și

dispatch (nume_proces_2): ready → *running*

Dacă procesul aflat în execuție lansează o operație de I/E înaintea expirării intervalului de timp ce i-a fost alocat, atunci el va trece în lista proceselor blocate, urmând ca primul proces din lista celor gata de execuție să fie lansat în execuție. Această stare de tranziție o precizăm astfel:

block (nume_proces_1): running → *blocked*

și

dispatch (nume_proces_2): ready → *running*

În cazul în care operația de I/E s-a terminat, procesul este trecut din starea blocat în starea gata de execuție. Starea de tranziție este:

wakeup (nume_proces_1): blocked → *ready*

Un proces ajunge în starea *suspendat* când are nevoie de o resursă pe care sistemul de operare nu i-o poate încă alocă, când invocă o rutină I/E sau așteaptă un semnal care nu s-a produs încă. Astfel de procese ies afară din competiția pentru UC până când condiția de suspendare dispăre. Sistemul de operare înregistrează motivul suspendării pentru a putea relua procesul mai târziu, când condiția de suspendare dispăre datorită acțiunii altor procese sau apariției unui eveniment extern. După apariția evenimentului, se poate întâmpla ca UC să nu execute acest proces, întrucât în sistem există un altul cu prioritate mai mare.

La un moment dat, procesele din sistem se pot afla în diverse stări, totalitatea acestora definind *starea globală a sistemului*. Ca răspuns la acțiunile interne sau externe, procesele își pot schimba repede starea rezultând a nouă stare globală a sistemului.

3.2.3. Planificarea unității centrale

Existența simultană în memorie a mai multor procese face posibil ca prin intermediul unui mecanism de planificare a UC, să se îmbunătățească eficiența globală a sistemului de calcul, realizându-se un volum mai mare de lucru într-un timp mai scurt².

Procesele încărcate în memorie și gata de a fi lansate în execuție sunt grupate într-un șir de așteptare (șir ready) în vederea alocării UC. Implementarea acestui șir (numit de multe ori “coadă” de așteptare) se realizează de obicei sub forma unei liste înlănțuite ale cărei elemente sunt blocurile de control asociate proceselor (BCP), fiecare BCP incluzând un indicator (pointer) către procesul care îi urmează în șirul ready. Deoarece pe lângă UC, procesele folosesc și alte resurse ale sistemului de calcul, pentru fiecare dintre aceste resurse pot exista șiruri de așteptare (“cozi”), numite, de exemplu, șiruri de dispozitiv sau șiruri I/E dacă este vorba despre procesele ce așteaptă eliberarea unui dispozitiv de I/E.

Ori de câte ori UC devine inactivă, sistemul de operare, prin intermediul unei

²În general, un proces folosește UC până în momentul în care trebuie să aștepte realizarea unei operații de I/E. Fără multiprogramare, pe durata realizării acestei operații UC rămâne inactivă. Cu multiprogramare însă, sistemul de operare permite imediat unui alt proces să folosească UC, eliminând timpul de așteptare din primul caz.

componente numite planificator (*scheduler*), selectează pentru execuție unul dintre procesele aflate în șirul ready. Planificatorul UC poate fi planificator pe termen lung (de perspectivă) sau planificator pe termen scurt (imediat) (figura 3.3).

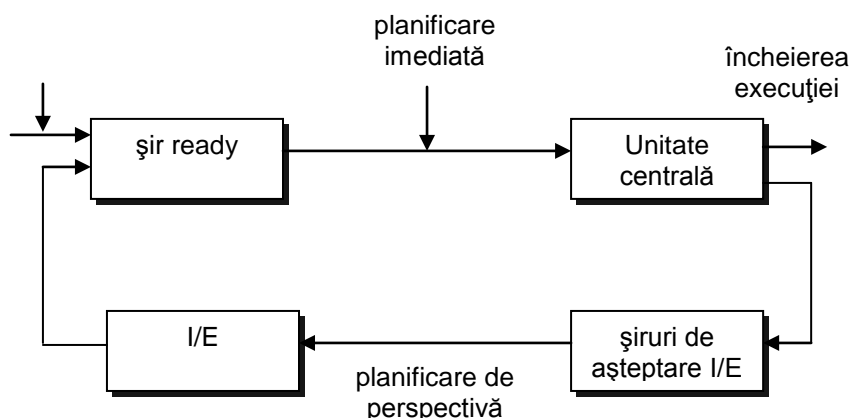


Fig. 3.3. Planificarea execuției proceselor

Planificatorul pe termen scurt (numit și planificator UC) selectează unul dintre procesele gata de execuție aflate deja în memoria internă și îi aloacă UC. Deoarece multe procese conțin cicluri “rafală” UC foarte scurte (câteva milisecunde), planificatorul pe termen scurt are o frecvență de execuție foarte mare și, prin urmare, trebuie să fie foarte rapid pentru a nu irosi timpul de lucru al UC.

Planificatorul pe termen lung (numit și planificator de job-uri) stabilește care sunt procesele ce vor fi încărcate în memoria internă a sistemului pentru a fi executate atunci când există mai multe cerere decât posibilitățile imediate de execuție. De obicei, în astfel de situații, procesele se află memorate pe disc magnetic, de unde planificatorul pe termen lung le selectează și le încarcă în memorie în vederea execuției. Cu alte cuvinte, planificatorul pe termen lung controlează gradul multiprogramării (numărul proceselor din memorie). În mod evident, frecvența de execuție a acestui tip de planificator este mult mai mică și el poate folosi mai mult timp, decât planificatorul pe termen scurt, pentru a selecta procesele. Există și sisteme care folosesc un nivel suplimentar de planificare prin intermediul unui planificator pe termen mediu (vezi figura 3.4).

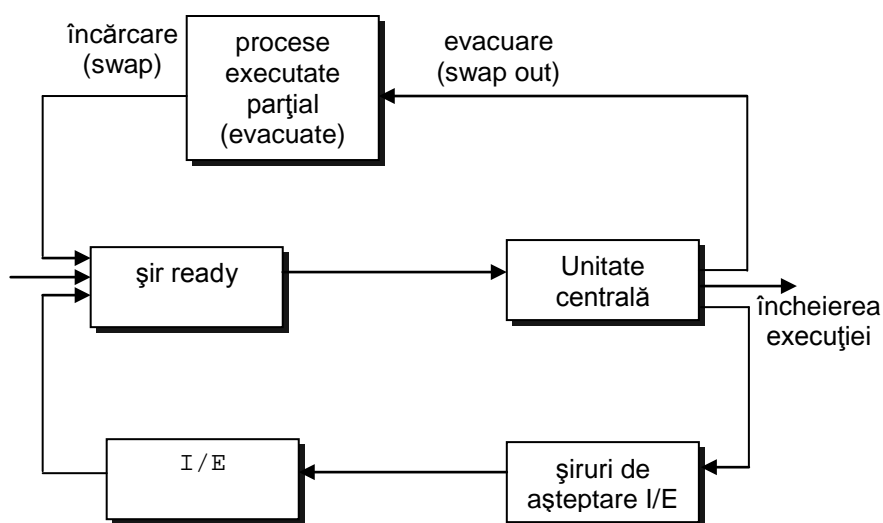


Fig. 3.4. Planificare pe termen mediu

Rolul acestuia este de a modifica gradul de multiprogramare atunci când este necesar, evacuând din memorie anumite procese (care altfel ar concura pentru dobândirea UC) și reintroducându-le în momentul în care încărcarea sistemului permite reluarea execuției din

punctul în care fusese întreruptă. Pentru această metodă se folosește în limbajul de specialitate denumirea de *swapping* ceea ce ar însemna interschimbare sau, mai sugestiv, evacuare și introducere din/în memorie a procesului. Planificatoarele pe termen mediu sunt incluse mai ales în cadrul sistemelor de tip time-sharing și a celor cu memorie virtuală.

3.2.3.1 Algoritmi de planificare a UC

Prezentarea cât mai fidelă a funcționării algoritmilor de planificare UC ar presupune utilizarea unor exemple care să includă un număr mare de procese, fiecare dintre ele fiind o secvență de sute de cicluri “rafală” UC și I/E. Pentru ca toate acestea să nu afecteze înțelegerea elementelor esențiale, s-a preferat folosirea unei variante simplificate în care fiecare proces conține un singur ciclu “rafală” UC, iar performanțele sunt apreciate cu ajutorul duratei medii de așteptare (notată DMA).

◇ Algoritmul FCFS

Algoritmul FCFS (First Come First Served) este cel mai simplu algoritm de planificare a UC: primul proces care cere alocarea UC este cel care o obține. Șirul ready este de tip FIFO (First In First Out); atunci când un nou proces devine gata de execuție, blocul său de control se adaugă la sfârșitul șirului ready; ori de câte ori devine disponibilă, UC este alocată procesului aflat pe prima poziție a șirului.

Dacă în sistem există următoarele procese, sosite la același moment de timp în ordinea numerotării:

Proces	Durata ciclului “rafală”
1	10
2	29
3	3
4	7
5	12

algoritmul de planificare a UC de tip FCFS va determina executarea proceselor conform diagramei din figura următoare:

Proces 1	Proces 2	Proces 3	Proces 4	Proces 5
0	10	39	42	49

Duratele de așteptare impuse fiecărui proces până la ocuparea UC sunt:

Proces	Durata de așteptare
1	0
2	10
3	39
4	42
5	49

Durata medie de așteptare este deci $(0 + 10 + 39 + 42 + 49) : 5 = 28$. Se observă că dacă procesele ar fi sosit în altă ordine (de exemplu 3, 4, 1, 5, 2), execuția lor ar fi avut evoluția din figura următoare:

Proces 3	Proces 4	Proces 1	Proces 5	Proces 2
0	3	10	20	61

durata medie de așteptare fiind mult mai mică: $(0 + 3 + 10 + 20 + 32) : 5 = 13$

Exemplul ales ilustrează unul dintre dezavantajele algoritmului FCFS: durata medie de așteptare, în general, nu este minimală și poate varia între limite foarte largi, în funcție de caracteristicile proceselor implicate în planificare. În plus, dacă încărcarea sistemului nu este echilibrată poate să apară așa-numitul *efect de convoi*.

Presupunând că în sistem există un proces limitat UC și mai multe procese limitate I/E, se poate întâmpla ca la un moment dat să se aloce UC procesului limitat UC. În timp ce acesta se execută, toate celelalte procese își încheie operațiile de I/E și intră în șirul ready, în așteptarea eliberării UC. Pe durata așteptării, dispozitivele de I/E stau nefolosite. În momentul în care procesul limitat UC își încheie ciclul “rafală” și trece într-un șir de așteptare I/E, toate procesele limitate I/E (care au cicluri “rafală” UC foarte scurte) se execută rapid și intră pe rând în șirurile de așteptare I/E. Este deci rândul UC să stea nefolosită. Apoi ciclul de funcționare prezentat se reia, generând o utilizare inefficientă atât a UC cât și a dispozitivelor de I/E.

◇ Algoritmul SJF

Algoritmul SJF (Shortest Job First - se execută mai întâi cel mai scurt job) ia în considerare pentru fiecare proces următorul ciclu “rafală” UC pe care îl conține și alocă UC (atunci când ea devine disponibilă) procesului cu cel mai scurt ciclu “rafală” UC următor existent în șirul ready. În cazul în care există două procese cu aceeași durată a ciclului “rafală” UC următor, între ele se aplică regula FCFS.

Ca exemplu, se poate folosi același set de procese utilizat în cazul algoritmului FCFS în care procesele soseau în sistem la același moment de timp, în ordinea numerotării:

Proces	Durata ciclului “rafală”
1	10
2	29
3	3
4	7
5	12

Algoritmul de planificare a UC de tip SJF, examinând durata ciclului “rafală” asociat fiecărui proces, va planifica execuția conform figurii anterioare, rezultând o valoare a duratei medii de așteptare de 13 unități de timp. Pentru comparație, se poate observa că algoritmul FCFS, folosind același set de procese genera o durată medie de așteptare de 28 de unități de timp.

Se poate demonstra că planificând un proces scurt înaintea unuia lung, durata de așteptare a procesului scurt se micșorează cu mai mult decât crește durata de așteptare a procesului lung și, prin urmare, durata medie de așteptare se reduce în mod substanțial. Din acest motiv, algoritmul SJF este optimal, asigurând o durată medie de așteptare minimă, oricare ar fi setul de procese luat în considerare. Singura problemă care apare este legată de cunoașterea duratei ciclului “rafală” UC următor asociat fiecărui proces analizat.

Dacă se dorește planificarea pe termen lung într-un sistem de tip cu prelucrare pe loturi (*batch*), se folosește ca valoare durată limită a job-ului precizată de către fiecare utilizator în parte³. Dacă însă se dorește folosirea algoritmului pentru planificarea pe termen scurt, deoarece nu se pot cunoaște cu exactitate duratele ciclurilor “rafală” următoare ale proceselor implicate, se poate realiza doar o aproximare a funcționării reale.

◇ Algoritmi bazați pe priorități

În cadrul acestui tip de algoritmi, fiecărui proces i se asociază o prioritate (reprezentată,

³În cazul în care utilizatorul estimează și precizează o valoare prea mică, poate să apară eroare de depășire a limitei de timp, fiind necesară replanificarea job-ului.

în general ca un număr cuprins într-o gamă fixată de valori), UC fiind alocată (în momentul în care devine disponibilă) procesului cu cea mai mare prioritate din șirul ready. SJF este un caz particular de algoritm bazat pe priorități în care prioritatea fiecărui proces este un număr invers proporțional cu mărimea ciclului “rafală” UC următor.

Prioritatea se poate defini intern sau extern. Atunci când este de tip intern, prioritatea procesului se calculează pe baza unei entități măsurabile cum ar fi, de exemplu, limita de timp, necesarul de memorie, numărul fișierelor deschise sau raportul dintre numărul mediu de cicluri “rafală” I/E și numărul mediu de cicluri “rafală” UC. Dacă definiția este de tip extern, criteriile folosite sunt din afara sistemului de operare: tipul și mărimea fondurilor rambursate pentru utilizarea calculatorului, departamentul care sponsorizează lucrarea, factorii politici, etc.

Principala problemă a algoritmilor bazați pe priorități este posibilitatea apariției blocării la infinit a proceselor care sunt gata de execuție, dar, deoarece au prioritate redusă nu reușesc să obțină accesul la UC⁴. O astfel de situație poate să apară într-un sistem cu încărcare mare în care se execută un număr considerabil de procese cu prioritate ridicată; acestea vor obține mereu accesul la UC în detrimentul proceselor cu prioritate redusă care este posibil să nu se mai execute niciodată.

O metodă de rezolvare a acestei probleme este “îmbătrânirea” proceselor, o tehnică prin care se mărește treptat prioritatea proceselor care se constată că rămân în sistem un timp mai îndelungat. De exemplu, dacă prioritățile sunt cuprinse în domeniul 0 până la 64, se poate stabili ca la fiecare 10 minute să fie incrementat cu câte o unitate prioritatea proceselor rămase în așteptare. În acest fel chiar și un proces cu prioritatea inițială 0 va reuși ca într-un timp destul de scurt să ajungă cel mai prioritar și, prin urmare să obțină accesul la UC (să se execute).

Algoritmi preemptivi

Toți algoritmi de planificare a UC descriși anterior (FCFS, SJP și algoritmi bazați pe priorități) sunt algoritmi ne-preemptivi: odată alocată UC ea este folosită de către proces până în momentul în care acesta dorește să o elibereze (își încheie execuția sau urmează să efectueze o operație de I/E). Un algoritm preemptiv permite însă întreruperea execuției unui proces în momentul în care în șirul ready apare un alt proces cu drept prioritar de execuție, sistemul de operare alocându-i imediat acestuia UC.

Algoritmul FCFS este prin definiție ne-preemptiv; ceilalți doi algoritmi prezentați pot fi modificați, astfel încât să devină preemptivi.

SJF preemptiv se formulează astfel: dacă în șirul ready sosește un proces al cărui ciclu “rafală” UC următor este mai scurt decât cea ce a mai rămas de executat din ciclul “rafală” UC al procesului curent, se întrerupe execuția celui din urmă și se alocă UC noului proces.

Un algoritm preemptiv bazat pe priorități funcționează într-un mod similar: ori de câte ori sosește în șirul ready un nou proces, prioritatea sa este comparată cu cea a procesului curent; dacă se constată că este mai mare, se întrerupe execuția procesului curent și se alocă UC procesului nou⁵.

Ca exemplu pot fi folosite următoarele trei procese, sosite în șirul ready la momentele de timp specificate:

Proces	Momentul sosirii în șirul ready	Durata ciclului “rafală”
1	0	12
2	3	3

⁴Fenomenul se mai numește și “înfometarea proceselor”.

⁵Dacă algoritmul bazat pe priorități este de tip ne-preemptiv, execuția nu se întrerupe, noul proces fiind pus la începutul șirului ready, conform priorității pe care o are și așteptând eliberarea UC.

3	4	6
---	---	---

Utilizând un algoritm de planificare de tip SJF fără preempție se obține o evoluție a execuției proceselor ca în figura următoare:

Proces 1	Proces 2	Proces 3
0	12	15
		21

durata medie de așteptare fiind: $(0 + (12-3) + (15-4)) : 3 = 6,67$

Dacă se folosește un algoritm SJF preemptiv, durata medie de așteptare devine: $(0 + (12-3) + (3-3) + (6-4)) : 3 = 3,67$ evoluția execuției fiind ilustrată în figura următoare:

Proces 1	Proces 2	Proces 3	Proces 1
0	3	4	6
			12
			21

Deoarece inițial în șirul ready există numai procesul 1, acesta se execută până în momentul 3, când apare în șir procesul 2, a cărui durată de ciclu "rafală" (3 unități de timp) este mai mică decât cea ce a mai rămas de executat din procesul 1 (9 unități de timp). Prin urmare, procesul 1 va fi întrerupt, UC fiind alocată procesului 2, care se va executa fără întrerupere deoarece la momentul 4, când în șirul ready sosește și procesul 3, necesarul duratelor de execuție se prezintă astfel:

Proces	Necesar
1	9
2	2
3	6

În momentul încheierii execuției procesului 2 va fi planificat mai întâi procesul 3 și abia după aceea procesul 1.

◇ Algoritmul Round-Robin

Round-Robin este un algoritm de planificare a UC proiectat special pentru sistemele time-sharing. Principalele sale caracteristici sunt definirea și folosirea unei cunute temporale (cu valori cuprinse în domeniul 10 ms până la 100 ms) și tratarea șirului ready ca șir FIFO circular. Planificatorul alocă pe rând UC fiecărui proces din șir pe o durată egală cu cel mult o cunată (se folosește un ceas setat în mod corespunzător).

Dacă durata ciclului "rafală" UC al procesului curent este mai mică decât durata cuantei, însuși procesul eliberează UC prin emiterea unei cereri de I/E sau prin comunicarea încheierii execuției. Dacă însă durata ciclului "rafală" UC depășește durata cuantei, ceasul va genera o întrerupere și procesul va fi inclus la sfârșitul șirului ready, după ce în prealabil contextul său a fost salvat în blocul de control asociat.

Se poate spune deci că algoritmul Round-Robin este un algoritm preemptiv, care asigură un timp aproape egal de așteptare pentru toate procesele din sistem. Într-adevăr, dacă în șirul ready există N procese și se lucrează cu o cuantă de valoare C, fiecare proces va folosi în total $\frac{1}{N}$ din timpul UC, fiecare alocare durând cel mult C unități de timp. Între două alocări succesive către același proces durata de așteptare va fi de cel mult $(N-1) \times C$ unități de timp.

Ca exemplu vom fi folosit același set de procese ca și în cazul algoritmului FCFS:

Proces	Durata ciclului "rafală"
1	10
2	29
3	3
4	7
5	12

Presupunând că se alege ca valoare a cuantei durata de 10 unități de timp, evoluția execuției proceselor planificate cu ajutorul algoritmului Round-Robin va fi cea din figura următoare:

Proces 1	Proces 2	Proces 3	Proces 4	Proces 5	Proces 2	Proces 5	Proces 2
0	10	20	23	30	40	50	61

Procesul 1 folosește complet prima cuantă, încheindu-și execuția. Procesul 2 având nevoie de 29 de unități de timp este întrerupt în momentul expirării celei de a doua cuante, UC fiind alocată următorului proces din șirul ready. Procesul 3 se încheie înainte de epuizarea cuantei curente, eliberând UC pentru procesul 4, care are o comportare asemănătoare. În final rămân în șirul ready numai procesele 5 și 2 care folosesc alternativ UC pe durata a încă două cuante complete și două cuante incomplete.

Exemplul prezentat permite compararea performanțelor algoritmilor FCFS, SJF și Round-Robin. Durata medie de așteptare generată de planificarea setului de procese cu ajutorul algoritmului Round-Robin este: $(0 + (10+20+2) + 20 + 23 + (30+10)) : 5 = 23$ unități de timp

Reamintim că, pentru același set de procese, algoritmul FCFS generează 28 unități de timp, iar algoritmul SJF doar 13 unități de timp. Se observă că algoritmul Round-Robin asigură o valoare intermediară, în timp ce SJF este în mod evident cel mai performant.

Performanțele algoritmului Round-Robin depin în mod esențial de mărimea cuantei folosite.

◇ Șiruri de procese multinivel

Atunci când procesele existente în sistem pot fi clasificate în grupe diferite, în funcție de anumite caracteristici (de exemplu: valori ale timpului de răspuns, prioritate definită extern, necesar de memorie, etc.), se utilizează un algoritm de planificare pentru șiruri multinivel.

Șirul ready este format din mai multe subșiruri, fiecare dintre acestea conținând câte o categorie de procese și având propriul algoritm de planificare. De exemplu, se pot crea două subșiruri: unul pentru procese interactive (*foreground*) și altul pentru procese de tip batch (*background*); pentru primul se poate folosi un algoritm de tip Round-Robin, iar pentru cel de-al doilea un algoritm FCFS. Este important de reținut faptul că și între subșiruri trebuie să existe un algoritm de planificare (de cele mai multe ori un algoritm preemtiv bazat pe priorități nemodificabile). De exemplu, se poate stabili ca subșirul proceselor interactive să aibă prioritate mai mare decât cel al proceselor de tip batch, ceea ce va face ca procesele din al doilea subșir să se poată executa numai atunci când primul șir este vid; dacă între timp apare un proces în primul șir, se întrerupe execuția procesului curent și UC este alocată noului sosit. O altă variantă de planificare între subșiruri este stabilirea a câte unei perioade de timp în care fiecare dintre acestea să dețină UC în mod exclusiv. De exemplu, pentru subșirul proceselor interactive se poate acorda 80% din timpul total al UC, iar pentru subșirul proceselor de tip batch restul de 20%.

◇ Șiruri de procese multinivel cu feedback

Spre deosebire de algoritmul prezentat anterior, în care procesele, odată introduse într-un subșir, rămâneau în cadrul acestuia până la încheierea execuției, în cazul unui algoritm de

planificare pentru șiruri multinivel cu feedback, se permite mutarea unui proces dintr-un subșir în altul, în funcție de anumite caracteristici dinamice. Metoda favorizează procesele interactive și procesele limitate I/E care sunt plasate în subșiruri cu prioritate ridicată, “retrogradând” procesele care folosesc UC un timp prea îndelungat în subșiruri cu prioritate redusă. De asemenea, ea previne apariția fenomenului de “înfometare” (blocare la infinit) printr-o formă de “îmbătrânire”: permite proceselor care așteaptă de prea mult timp în șirurile de prioritate redusă să “promoveze” în cadrul șirurilor de prioritate ridicată.

Pentru a putea defini complet un algoritm de planificare pentru șiruri multinivel cu feedback este necesară precizarea mai multor parametri:

- numărul de subșiruri;
- algoritmul de planificare asociat fiecărui subșir;
- metoda de stabilire a subșirului în care intră procesul atunci când dorește să se execute;
- criteriul și metoda de “promovare” a unui proces într-un subșir cu prioritate ridicată;
- criteriul și metoda de “retrogradare” a unui proces într-un subșir cu prioritate redusă.

Având un înalt grad de generalitate, algoritmul de planificare pentru șiruri multinivel cu feedback este cel mai complex, oferind avantajul unor performanțe ridicate, dar necesitând în același timp un număr mare de informații pentru stabilirea valorilor optime în cazul parametrilor de configurare menționați anterior.

3.3. GESTIUNEA MEMORIEI

3.3.1 Gestiunea în cazul multiprogramării

Organizarea memoriei cu partiții fixe

Cea mai simplă metodă pentru administrarea memoriei în acest caz este de a o împărți în n partiții fixe, posibil inegale (mecanism de alocare static). O partiție este alocată unui proces pe toată durata execuției lui, indiferent dacă o ocupă complet sau nu. Când sosește un proces, el va fi pus în coada formată pentru cea mai mică partiție suficient de mare pentru a-l cuprinde.

Deoarece partițiile sunt fixe (au lungimi prestabilite, nemodificabile), orice spațiu neocupat de un proces dintr-o partiție va fi pierdut. Se produce astfel o *fragmentare internă a memoriei* (figura 3.5.):

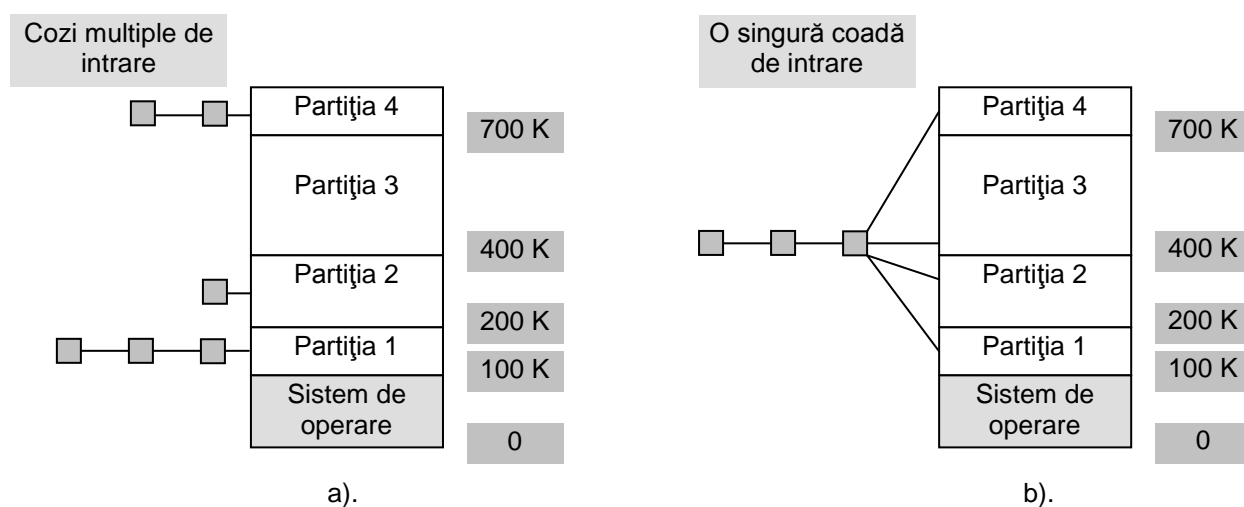


Fig. 3.5. Organizarea memoriei cu partiții fixe

În general există două *moduri de legare* a proceselor la partiții :

- fiecare partiție are o coadă proprie - operatorul stabilește de la început care sunt procesele ce vor fi executate în fiecare partiție;

- o singură coadă pentru toate partițiile - sistemul de operare alege, pentru procesul care urmează să intre în lucru, în ce partiție se va executa.

Dezavantajul prezentat în figura 3.5.a) - coada pentru partiții mari este goală (rezultă partiție liberă, posibil de a fi alocată, dar nu există cerere), iar cea pentru partiții mici este plină.

În cazul 3.5 b), ori de câte ori o partiție devine liberă, primul proces din coadă care încapă în acea partiție este încărcat și rulat. Deoarece nu se dorește folosirea unei partiții mari pentru un proces mic (pierdere de spațiu), o alternativă ar fi căutarea în întreaga coadă a celui mai mare proces care se potrivește partiției devenite libere. Această alternativă are dezavantajul că proceselor mici (de obicei interactive) li se acordă cel mai prost serviciu și nu cel mai bun, cum ar fi de dorit.

O soluție este aceea de a avea cel puțin o partiție mică ce permite proceselor mici să ruleze fără să aștepte alocarea unei partiții mai mari.

O altă soluție ar fi ca un proces ce poate rula să nu fie omis de mai mult de k ori, de fiecare dată când este omis, un contor asociat lui este incrementat; iar când a ajuns la valoarea k , el nu va mai fi omis la următoarea căutare în coadă.

Multiprogramarea ridică câteva probleme esențiale ce trebuie rezolvate (indiferent de modul de organizare a memoriei): *relocarea*, *protecția* și *partajarea*.

Discutăm aceste probleme relativ la organizarea memoriei în partiții fixe.

Relocarea. Din figură se observă că procese diferite pot rula la adrese diferite, depinzând de partiția în care este încărcat. Dacă, de exemplu, prima instrucțiune este un apel de procedură la o adresă relativă, adresa absolută se obține prin adunarea adresei relative cu 100 KB, dacă programul este încărcat în prima partiție, respectiv cu 200 KB, dacă este încărcat în a doua partiție, etc. O soluție ar fi de a modifica adresa relativă în adresa absolută la încărcarea programului (se cunoaște adresa de început a partiției). Pentru aceasta, trebuie inclus în codul programului executabil o listă sau o hartă de biți care să specifice care cuvinte din program sunt adrese (ce trebuie relocate) și care nu (sunt coduri de operații, constante, etc.).

Relocarea în timpul încărcării programului (numită și *relocare statică*) nu rezolvă problema protecției. Astfel, un program poate să acceseze în scriere sau citire orice cuvânt din memorie.

O soluție care rezolvă relocarea este de a echipa calculatorul cu un registru special, numit *registru de bază* (conținutul lui este păstrat în BCP-ul procesului). În acest caz avem de a face cu *relocarea dinamică*.

Astfel, când un proces este ales să ruleze, registrul de bază este încărcat cu adresa de început a partiției. Orice adresă de memorie generată de program este adunată cu conținutul registrului de bază.

Un avantaj oferit de utilizarea acestui registru este acela că mutarea programului în memorie în timpul rulării presupune doar schimbarea valorii din registrul de bază. În cazul relocării în timpul încărcării, mutarea unui program presupune reluarea procesului de generare a adreselor absolute.

Protecția. În sistemele multiutilizator, nu se acceptă ca procesele unui utilizator să scrie sau să citească datele aparținând altui utilizator.

Problema se pune atât la protecția proceselor între ele cât și la protecția spațiului de memorie alocat sistemului de operare, de accese neautorizate.

În sistemele ce folosesc registrul de bază pentru relocare, se utilizează un alt registru hardware numit *registru limită*, al cărui conținut este de asemenea păstrat în BCP-ul procesului, și care se încarcă cu lungimea partiției. Orice adresă generată (cu ajutorul registrului de bază) este verificată să nu depășească partiția curentă.

Partajarea. Pe lângă protecție, un bun mecanism de gestionare a memoriei trebuie să asigure partajarea datelor și codului între procesele care cooperează. Probabil că cel mai simplu mod de implementare a partajării, fără a compromite protecția, este de a lăsa în seama sistemului de operare controlul accesului la resursele partajate. Această soluție are unele dezavantaje: creșterea codului sistemului de operare precum și dificultatea protejării obiectelor create

dinamic.

O soluție este de a copia obiectul partajat în spațiul privat (partiția) al fiecărui proces ce îl partajează. Astfel, protecția este asigurată. Ca urmare, actualizările se realizează doar pe copiile obiectului, nu și pe obiectul însuși. De asemenea, aceste actualizări trebuie propagate și spre celelalte copii. De obicei, la fiecare comutare de context, sistemul de operare copiază datele partajate din spațiul de adrese al procesului curent activ, în spațiul de adrese al celorlalte procese participante.

Dezavantaje: menținerea în memorie a mai multor copii; în cazul partajării codului, copierea lui nu are nici un sens deoarece nu are loc nici o modificare.

O altă soluție este de a plasa datele într-un loc comun, dedicat acestui scop. În acest caz se pune problema rezolvării protecției: orice acces dincolo de partiția alocată este privit ca o violare a protecției. Cum se rezolvă:

În sistemele cu chei de protecție, la fiecare comutare de context se schimbă cheile tuturor blocurilor partajate cu scopul de a acorda drepturi de acces procesului curent activ. Aceasta presupune ținerea unei evidențe a blocurilor, care sunt partajate și de către cine, precum și a frecvențelor modificării ale cheilor.

În sistemele ce folosesc regiștrii de bază și limită, avem nevoie de seturi diferite de perechi de regiștri dedicați spațiului de memorie privat și respectiv celui partajat. Aceasta implică existența unui anumit mijloc, preferabil automat, de a desemna care este setul potrivit de regiștri pentru fiecare referire la memorie.

Cele mai multe dezavantaje ale acestei metode de organizare a memoriei se datorează partiționării statice, sistemul devenind inflexibil, neputându-se adapta la schimbări.

O primă problemă este fragmentarea internă ce rezultă din diferența dintre dimensiunea partiției și dimensiunea procesului ce o ocupă.

Partiționarea fixă impune restricții severe dimensiunii programului: nici un proces nu are voie să depășească dimensiunea celei mai mari partiții. Astfel, ea devine ineficientă în sistemele în care programele își modifică dinamic structurile de date (heap și stivă). Un alt dezavantaj este gradul de multiprogramare limitat la numărul de partiții.

Organizarea memoriei cu partiții variabile

Un nou mod de organizare a memoriei care elimină multe dintre problemele create de partiționarea fixă a memoriei este organizarea memoriei cu partiții variabile (mecanism de alocare dinamic). În funcție de solicitări și de capacitatea de memorie încă disponibilă la un moment dat, numărul și dimensiunile partițiilor se modifică automat. Se îmbunătățește factorul de utilizare a memoriei dar se complică alocarea și dealocarea memoriei.

În figura 3.6. sunt prezentate mai multe stări succesive ale memoriei:

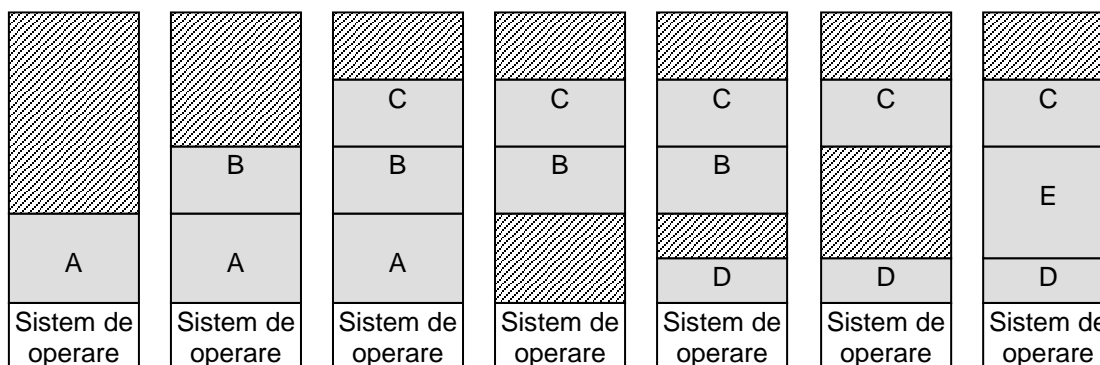


Fig. 3.6. Organizarea memoriei cu partiții variabile

Este ușor de observat că dacă sistemul funcționează timp îndelungat, atunci numărul spațiilor libere va crește, iar dimensiunile lor vor scădea, rezultând o fragmentarea externă a memoriei.

În momentul în care un proces nu are spațiu liber de memorie în care să se încarce, sistemul de operare poate lua una din următoarele trei decizii:

1. procesul așteaptă până când i se eliberează un spațiu suficient de mare de memorie;
2. sistemul de operare încearcă alipirea unor spații libere vecine, în speranța că va obține un spațiu de memorie suficient de mare (*cumularea golurilor*);
3. sistemul de operare decide efectuarea unei operații de compactare a memoriei (relocare), adică de deplasare a partițiilor active pentru a se absorbi toate zonele (fragmente) de memorie neutilizate (*compactarea*).

Fragmentarea memoriei poate să apară din mai multe motive, unul dintre ele fiind faptul că un program și-a terminat execuția, iar locul rămas liber prin ieșirea acestuia din memorie nu poate fi ocupat de un alt program. Golurile în utilizarea memoriei astfel rezultate pot fi folosite de alte programe cu dimensiuni mai mici sau cel mult egale cu cele ale golului. Într-o astfel de situație dacă un program a încăput într-o partiție astfel eliberată este posibil ca el să nu ocupe toată zona de memorie aferentă partiției, rezultând o nefolosire optimă a memoriei. Se impune deci realizarea unei operații de colectare a golurilor astfel apărute, adică două sau mai multe goluri adiacente să se cumuleze într-un singur gol de dimensiune mai mare. Acest proces se numește *cumularea golurilor*, și el este ilustrat în figura 3.7.

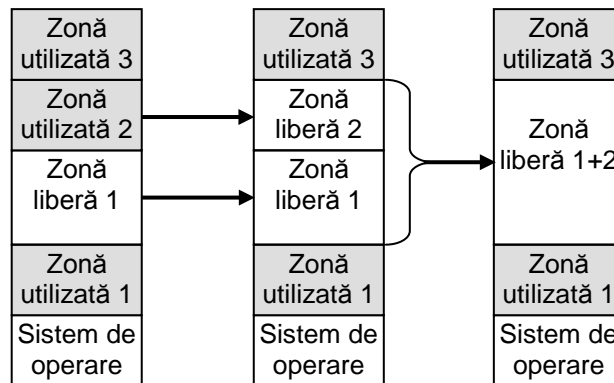


Fig. 3.7. Cumularea golurilor

Chiar în cazul în care golurile sunt colectate totuși este posibil, ca în cadrul memoriei interne, să apară goluri dispersate. Și în această situație poate apărea un job care să nu încapă în nici unul din golurile prezente. Job-ul respectiv s-ar putea rula totuși dacă toate golurile ce sunt în mod normal dispersate ar fi cumulate într-unul singur. Realizarea acestui proces se numește *compactarea memoriei*, o astfel de situație fiind ilustrată în figura 3.8.

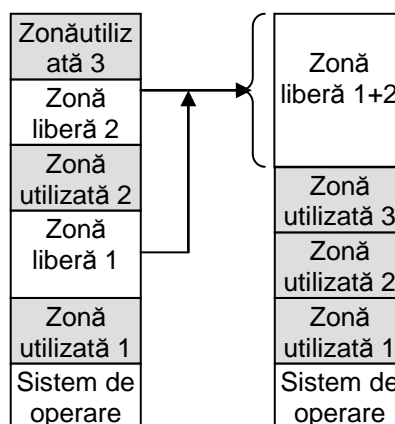


Fig. 3.8. Compactarea memoriei

De regulă, compactarea este o operație costisitoare și de aceea în practică se aleg soluții de compromis, cum ar fi :

- se lansează periodic compactarea (de exemplu, la 10 sec.), indiferent de starea sistemului. În intervalul dintre compactări, memoria apare ca un mozaic de spații ocupate care alternează cu spații libere. Procesele ce nu au loc în memorie așteaptă compactarea sau terminarea altui proces;
- se realizează o compactare parțială pentru a asigura loc numai procesului care așteaptă;
- se realizează compactarea ori de câte ori se eliberează o partiție.

Deși această schemă este cea mai bună din punct de vedere al eficienței, totuși ea prezintă următoarele inconveniente:

- consumarea resurselor calculatorului cu mutarea programelor dintr-o zonă în alta a memoriei;
- sistemul s-ar putea opri în timp ce se execută operația de compactare, ceea ce ar avea consecințe grave pentru programele utilizator;
- compactarea consumă foarte multă memorie secundară;
- compactarea mărește timpul de obținere a rezultatelor rulării programelor utilizator.

Protecția și partajarea.

Protecția și partajarea, nu diferă semnificativ față de cazul partiționării fixe. O diferență este aceea că se permite ca partiții adiacente să se suprapună. Astfel, o singură copie a unui obiect partajat poate fi accesibil din două spații de adresă diferite. Partajarea, însă, este limitată la două procese. În cazul în care sunt în joc mai multe procese, trebuie aplicate metodele de la partiționarea fixă.

Partajarea codului impune ca acesta să fie reentrant sau executat în manieră strict exclusivă, fără posibilitatea întreruperii de către alte procese. Reentranta presupune ca variabilele să fie păstrate în stivă sau în regiștri astfel încât, în cazul întreruperii procesului curent, să nu fie afectată starea procesului întrerupt.

Dintre avantajele acestei metode amintim:

- un proces poate ocupa întreaga memorie, în cazul în care îi este necesară (nu doar partiția cea mai mare, dacă are loc);
- în cazul extinderii unui proces dincolo de partiția pe care o ocupă, sistemul de operare poate crea o partiție mai mare și poate muta procesul în ea, sau partiția alocată procesului se poate extinde folosind zonele libere adiacente.

Partiționarea dinamică nu este lipsită de dezavantaje. Ea consumă mult timp și spațiu. Fragmentarea externă poate deveni o problemă serioasă, ducând la creșterea timpului în vederea compactării. Partajarea poate ridica, de asemenea, probleme dacă obiectele partajate se supun compactării.

Se poate realiza un compromis și anume: o parte din memorie să fie împărțită în partiții fixe. Nucleul sistemului de operare-ului, driverele de dispozitive și porțiuni ale sistemului de fișiere sunt buni candidați pentru partiționarea fixă. Restul memoriei poate fi alocată altor procese utilizând partiționarea variabilă.

3.3.2. Strategii de administrare a spațiului din memoria internă

Categorii de strategii:

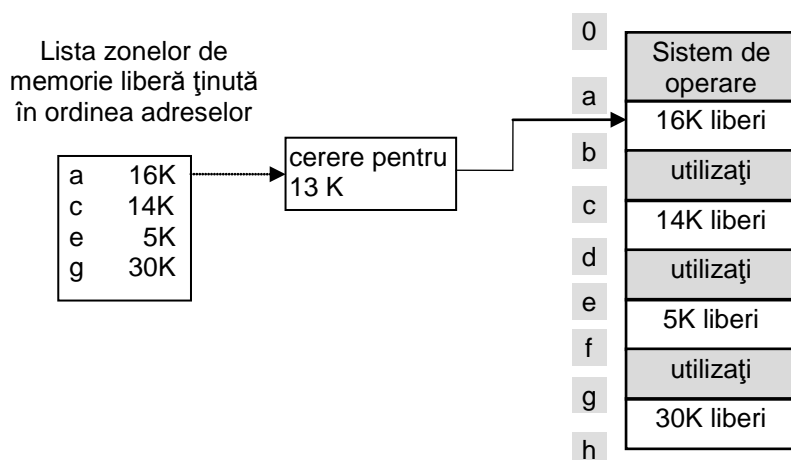
- **strategii de apel** - se bazează pe determinarea momentului când trebuie obținută următoarea porțiune din program sau de date pentru a o transfera din memoria secundară în memoria reală. Mult timp s-a folosit *strategia de apel la cerere* în care următoarea parte de program sau de date era încărcată în memorie la momentul în care programul aflat în execuție făcea referire la aceasta. În prezent, se tinde către utilizarea unor strategii anticipatorii, care să prevadă momentul de timp la care programul aflat în execuție va avea nevoie de o anumită porțiune de program din

memoria secundară pentru a o încălca din timp în memoria principală, îmbunătățind în acest fel performanțele sistemului.

- **strategii de plasare în memorie** - în general, în memorie se află “risipite” mai multe zone nealocate de diferite dimensiuni. Atunci când apare un job care necesită memorie, este aleasă una dintre acestea, cu dimensiune suficient de mare pentru a satisface cerințele job-ului. Dacă zona este prea mare, o parte se alocă job-ului, iar restul se integrează setului de zone disponibile. În momentul terminării execuției, job-ul eliberează zona de memorie aferentă, care va fi inclusă și ea în set. Dacă zona de memorie disponibilizată este adiacentă celorlalte, ele se pot contopi pentru a forma o zonă de dimensiune mare, moment în care se poate verifica dacă nu cumva există job-uri aflate în așteptare care ar putea folosi această nouă zonă. Situația prezentată este un caz particular al problemei de alocare dinamică a memoriei (care are ca obiect modul în care se poate satisface o cerere de dimensiune precizată având la dispoziție o listă de zone de memorie disponibile), pentru care există mai multe variante de rezolvare. Deci, strategiile de plasare în memorie se concentrează asupra determinării locului în care se va încălca un program în memoria primară.
- **strategii de înlocuire** - se ocupă cu determinarea acelei părți din program ce va fi scoasă din memorie pentru a face loc noului program ce intră în execuție.

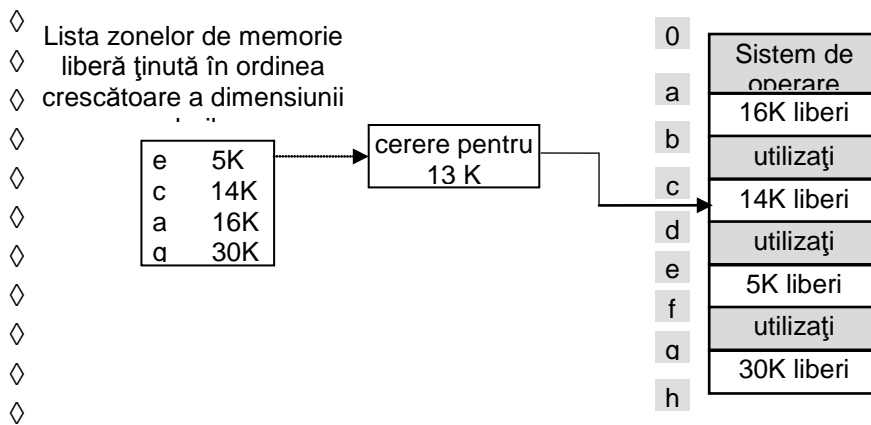
În cazul păstrării listei zonelor de memorie disponibile, ordonate după adresă, avem următorii algoritmi de alocare a memoriei (*strategii de plasare în memorie*) pentru un proces nou creat sau adus de pe disc :

First Fit (*prima potrivire*) - în cadrul acestei strategii, job-ului care vine să se execute i se va aloca prima zonă de memorie disponibilă cu dimensiune suficient de mare în care acesta poate să încapă. Căutarea zonei respective se poate face pornind fie de la începutul listei zonelor de memorie liberă, fie din oricare alt loc în care s-a terminat o căutare anterioară de același tip.

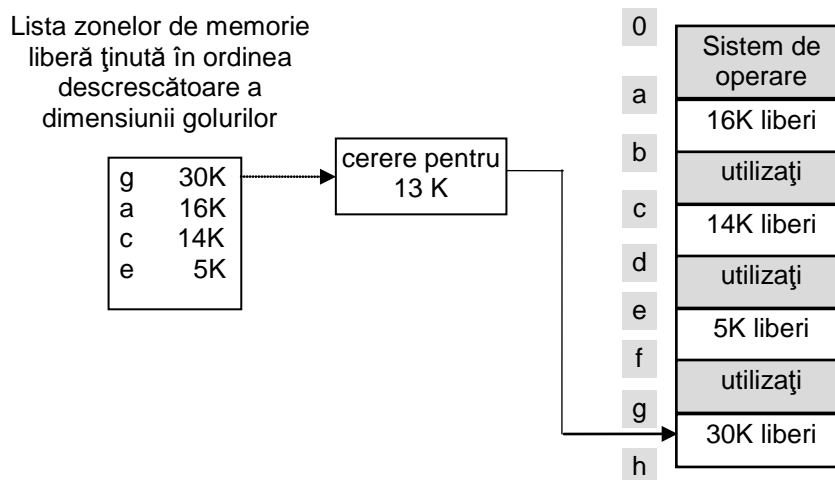


Next Fit (*următoarea potrivire*) – strategia lucrează la fel ca **First Fit**, dar căutarea pornește de la elementul alocat în potrivirea anterioară, nu de la capătul listei.

Best Fit (*cea mai bună potrivire*) - strategia alege, dintre toate zonele de memorie disponibile a căror dimensiune permite alocarea job-ului, zona cu cea mai mică dimensiune ce poate să-l încapă, minimizându-se astfel fragmentarea internă. Lista zonelor de memorie liberă este ordonată crescător după dimensiunile golurilor.



Worst Fit (*cea mai rea potrivire*) - în cazul acestei strategii se alocă job-ului zona de memorie disponibilă care are cea mai mare dimensiune. Lista zonelor de memorie liberă este ordonată descrescător după dimensiunea golurilor.



Quick Fit (*potrivire rapidă*) - se păstrează liste diferite pentru diverse dimensiuni ce pot fi cerute. De exemplu, putem avea un tablou cu n intrări în care prima intrare este un pointer spre capul listei spațiilor de 4K, a doua intrare spre lista de spații de 8K, a treia spre cele de 12K, etc. Cu acest algoritm, căutarea spațiului de dimensiune cerută este foarte rapidă, dar se pierde mult timp pentru găsirea spațiilor adiacente libere pentru colaționare (altfel, memoria se umple de spații mici, neutilizabile).

3.3.3. Mecanismul de swapping

În mod normal, sunt mai mulți utilizatori și deci mai multe procese decât pot fi păstrate în memoria principală și de aceea este necesar păstrarea proceselor în exces, pe disc (memoria secundară). Pentru rularea acestor procese, ele trebuie aduse în memoria principală. Această mutare a proceselor pe și de pe disc se numește **swapping**.

În principiu, swapping-ul se poate aplica și în cazul organizării memoriei în partiții fixe dar în general, acest mecanism se asociază organizării memoriei folosind partiții variabile.

O problemă o reprezintă dimensiunea spațiului de memorie ce trebuie alocat unui proces atunci când acesta este creat sau adus din nou în memorie prin swapping. Dacă procesele create își păstrează dimensiunea pe toată durata execuției, alocarea este foarte simplă: se alocă exact spațiul necesar, nici mai mult, nici mai puțin.

În timpul execuției, însă, segmentul de date poate crește, de exemplu, prin alocare dinamică a memoriei din heap. Dacă există un spațiu liber adiacent procesului curent, acesta îi poate fi alocat. Dacă nu, fie trebuie mutat procesul într-un spațiu suficient de mare, fie unul sau mai multe procese trebuie salvate pe disc pentru crearea acestui spațiu. Dacă un proces nu are loc

în memorie pentru extindere iar spațiul de swap de pe disc este plin, procesul este pus în starea de așteptare sau este distrus.

Dacă este de așteptat ca majoritatea proceselor să crească în timpul execuției, o idee bună este aceea de a alocă un mic extraspațiu ori de câte ori procesul este adus în memorie de pe disc sau este mutat. În momentul salvării pe disc, doar zona de memorie utilizată va fi supusă swapping-ului.

Ori de câte ori planificatorul trebuie să lanseze în execuție un proces și nu este suficient loc în memorie, va apela procesul de swapping. Procesele ce se supun swapping-ului se vor alege dintre procesele suspendate care ocupă un spațiu ce este suficient de mare pentru a fi alocat procesului curent.

În cazul operației de swapping pe disc, se vor salva locațiile ce conțin datele procesului, stiva și regiștrii UC relevanți, precum și codul, în cazul sistemelor ce permit modificarea codului în timpul execuției.

În ceea ce privește spațiul alocat pe disc swapping-ului, există două opțiuni :

- toate procesele salvate pe disc se vor depune într-un unic fișier de swap;
- fiecare proces va fi salvat separat.

În ambele situații, spațiul de swap corespunzător fiecărui proces din memorie se alocă static, la crearea procesului.

O altă problemă este dacă legarea proces-partiție se face static sau dinamic, adică dacă un proces salvat pe disc în urma unei operații de swap, va fi readus în memorie în aceeași partiție pe care o ocupa anterior swapping-ului sau în altă partiție. Legarea statică se recomandă în cazul partiționării statice a memoriei, eliminându-se astfel timpul afectat de către sistemul de operare alocării unei alte partiții. Pe de altă parte, sistemele unde procesele nu sunt legate permanent de o partiție specifică sunt mai flexibile și duc la o utilizare mai bună a memoriei.

3.4. SISTEMUL DE OPERARE UNIX

3.4.1. Elemente introductive privind sistemul de operare UNIX

Sesiunea de lucru sub UNIX

După pornirea sistemului de calcul, și încărcarea sistemului de operare, inițierea unei sesiuni de lucru sub UNIX începe cu acțiunea de conectare (`login`). În cadrul acesteia, utilizatorului `i` se va solicita, în primă fază, identificatorul de conectare unic (`login ID`) asociat. Acesta identifică utilizatorul și orice caracteristici asociate cu el, fiind în fapt un nume de recunoaștere autorizat. După furnizarea de către utilizator a identificatorului respectiv, `i` se va solicita și parola de acces corespunzătoare (`password`). Cele două elemente sunt verificate, și în cazul în care sunt corecte, conectarea se realizează, fiind afișate o serie de informații referitoare la ultima conectare nereușită, ultima conectare reușită, dacă sunt mesaje de poștă electronică, mesaje de la administratorul de sistem, ș.a. Apoi se lansează automat în execuție interpretorul de comenzi implicit (`shell`)⁶ afișându-se un prompter⁷. Din acest moment utilizatorul are acces la resursele sistemului, și poate folosi comenzile UNIX .

⁶Interpretorul de comenzi este un program executabil situat, de regulă, în directorul `/bin`, și care este tratat de către nucleul (kernel) sistemului de operare, ca orice proces utilizator neprivilegiat. La deschiderea unei sesiuni de lucru, se lansează în execuție un proces shell la terminalul la care lucrează utilizatorul, așteptând comenzi.

Interpretorul de comenzi (`shell`) (care pornește după introducerea numelui utilizatorului și a parolei) poate fi ales de către utilizator. Există mai multe interpretoare clasice, fiecare răspunzând anumitor cerințe. Interpretorul "standard" în UNIX este Bourne shell (`sh`), dar foarte folosite sunt și Bourne Again shell (`bash`), Korn shell (`ksh`), Berkeley C shell (`csh`), Turbo C shell (`tcsh`).

⁷În funcție de shell-ul utilizat, prompter-ul este reprezentat printr-un caracter specific. Astfel, în cazul shell-ului `sh` (Bourne shell), prompter-ul este `$` (în cazul utilizatorilor neprivilegiați) sau `*` (pentru utilizatori privilegiați - superuser, root). Dacă shell-ul este `csh` (C shell), prompter-ul va fi `%`, iar în cazul shell-ului `ksh` (Korn shell) prompter-ul va fi `:`.

Terminarea unei sesiuni de lucru UNIX se face folosind una din comenzile: `exit`, respectiv `bye`, sau tastând combinația de taste `<Ctrl><D>`.

Pentru oprirea, într-o manieră ordonată, a calculatorului pe care rulează sistemul UNIX, se utilizează comanda `shutdown`.

Aceasta poate fi executată numai de către utilizatorul privilegiat (`superuser`, `root`). Când comanda este executată, toți utilizatorii primesc un mesaj `shutdown` și apoi recepționează un mesaj final de terminare.

Câțiva dintre indicatorii utilizați cu comanda `shutdown` sunt:

- h oprește sistemul complet
- i permite afișarea de mesaje care ghidează utilizatorul în cadrul procesului de `shutdown`
- k simulează un `shutdown` sistem
- m oprește sistemul și îl trece în modul întreținere (`single user`), astfel încât administratorul de sistem să poată efectua întreținerea acestuia (instalarea de hardware/software sau întreținerea planificată a hardware-ului/software-ului existent)
- r permite oprirea sistemului urmată de restartare (`reboot`).

Este de asemenea posibilă specificarea momentului de timp la care oprirea (sau restartarea) să fie făcută, prin indicarea unei date viitoare sau a unui timp relativ. În acest caz sistemul trimite periodic mesaje utilizatorilor, referitoare la oprire.

Variantele cele mai des folosite ale acestei comenzi sunt:

```
shutdown -h now
shutdown -r now
```

Specificarea numelui fișierelor și directoarelor

UNIX are o structură de directoare și fișiere asemănătoare cu cea a sistemelor DOS și Windows.

Fișierele sunt stocate în directoare și sunt identificate printr-un nume, care este o secvență de caractere.

Toate numele de fișiere, oricare ar fi tipul lor (text, binare) se supun aceluiași reguli, și anume:

1. Spre deosebire de sistemul DOS unde numele de fișiere era specificat în formatul 8.3, (maxim 8 caractere pentru nume și maxim 3 caractere pentru o eventuală extensie), în UNIX se pot utiliza nume lungi de fișiere (până la 255 de caractere, dacă este instalat sistemul de fișiere `ext2` sau `umsdos`), iar numele pot conține mai mult de un punct (de exemplu, `mihai.stud.txt`);
2. În toate cazurile, în cadrul numelui unui fișier, UNIX face distincție între caracterele minuscule și majuscule⁸, (de exemplu, numele `FILENAME.tar.gz` și `filename.tar.gz` fac referire la fișiere diferite);
3. Caracterele ce pot fi utilizate pentru numele de fișiere sunt funcție de sistemul UNIX folosit, existând particularități. În general, pot fi utilizate literele de la A la Z sau de la a la z; numerele de la 0 la 9, linia de subliniere⁹, punctul¹⁰. În tabelul 4.1 sunt date o serie de caractere (sau combinații de caractere) ce au o semnificație specială pentru

⁸Este case-sensitive.

⁹Linia de subliniere poate separa cuvinte în cadrul numelui unui fișier, făcând numele mai ușor de citit. De exemplu, în loc de a numi un fișier `fișierdetest`, putem să-l numim `fișier_de_test`.

¹⁰Un punct poate fi folosit pentru a adăuga o extensie la un nume de fișier, într-o modalitatea similară fișierelor DOS. De exemplu, un fișier sursă C care conține un program numit `prog` poate fi denumit `prog.c`. Așa cum s-a arătat, în UNIX nu suntem limitați la o singură extensie. În cazul în care un punct este folosit ca prim caracter în cadrul numelui unui fișier, acesta conferă fișierului statutul de fișier ascuns (`hidden`). De exemplu, dacă în directorul curent există fișierele `doc1` și `.doc1`, tastând comanda `ls` va fi afișat numai fișierul `doc1`. Pentru afișarea ambelor fișiere se va tasta comanda `ls -a`.

UNIX, și ca urmare utilizarea lor trebuie făcută cu precauție. Lista nu este exhaustivă și ea depinde de shell-ul UNIX folosit.

Tabelul 3.1.

Semnificația câtorva caractere speciale

Caracter	Semnificație
\$	Indică începutul numelui unei variabile shell. De exemplu \$var va însemna numele unei variabile numite var.
	Leagă ieșirea standard a unei comenzi la intrarea standard a altei comenzi (pipe).
#	Începe un comentariu.
&	Execută un proces în fundal (background).
?	Specificator pentru nume de fișier global ce ține locul unui singur caracter.
*	Specificator pentru nume de fișier global ce ține locul mai multor caractere.
\$#	Numărul de argumente transmise unui script shell.
\$*	Argumente transmise unui script shell.
\$?	Returnează cod de la comanda executată anterior.
>	Operator pentru redirectarea ieșirii.
<	Operator pentru redirectarea intrării.
'	Substituție comandă.
>>	Operator pentru redirectarea ieșirii (pentru a adăuga la un fișier).
[]	Afișează un domeniu de caractere. [a-z] semnifică toate caracterele dintre a și z. [a,z] semnifică caracterele a sau z.
.nume_fisier	Execută fișierul cu numele specificat.
:	Separator de nume de director din cadrul căii.

Observații:

1. Sub UNIX, dacă un nume de fișiere conține spații (nu este recomandat, dar este posibil), ori de câte ori ne referim la el, va trebuie să includem numele respectiv între ghilimele duble.

2. Pentru fișierele executabile ("programe") nu există obligativitatea utilizării extensiilor: .COM, .EXE, .BAT. Aceste fișiere sunt marcate printr-un caracter * existent la sfârșitul numelui lor, atunci când sunt afișate cu comanda `ls -F`.

3. Sub DOS numele fișierelor arhivă (backup) au extensia .BAK, sub UNIX numele acestora se termină cu caracterul ~ (tilda).

Comenzi UNIX

În cadrul unui sistem UNIX utilizatorul are la dispoziție comenzi native UNIX precum și comenzi pe el însuși le poate scrie (fișiere de comenzi - fișiere script). În funcție de versiunea de UNIX ce rulează și de shell-ul utilizat, comenzile diferă.

Comenzile UNIX sunt de fapt programe executabile care pot fi găsite în directoarele /bin, /usr/bin. Diferențele între interpretoarele de comenzi se văd mai ales în contextul fișierelor de comenzi. Practic, aceste interpretoare permit scrierea de programe complexe, folosind comenzile UNIX și directivele speciale.

Forma sintactică generală a unei comenzi shell este următoarea:

Comanda[indicatori][argument1][argument2]..

unde:

comanda - reprezintă numele comenzii ce urmează a fi executată. Căutarea comenzii (a fișierului executabil), până când aceasta este găsită, se face în următoarea secvență: în directorul curent; în directorul /bin; în directorul /usr/bin. În cazul în care comanda nu este găsită,

va fi afișat un mesaj de eroare.

[indicatori] - reprezintă opțiuni desemnate printr-o literă, precedată de semnul - sau +, ce sunt destinate pentru a obține o rafinare a acțiunii comenzii. În cadrul unei comenzi pot fi specificați mai mulți indicatori, precedați de unul din cele două semne menționate anterior. De exemplu, comenzile `ls -a-l` și `ls -al` sunt echivalente.

[argument1] [argument2] .. - reprezintă nume de fișiere/directoare sau șiruri de caractere și constituie parametrii comenzii.

În funcție de comandă, argumentele pot fi opționale sau obligatorii. Specificarea argumentelor opționale, în cadrul sintaxei unei comenzi, se face prin introducerea lor între paranteze pătrate.

Specificarea argumentelor (fișierelor) este permisă de către `shell` și prin utilizarea următoarelor metacaractere (specificatori pentru nume de fișier global - wildcard):

* semnifică orice șir de caractere, inclusiv șirul vid

? semnifică orice caracter

[...] semnifică o mulțime de caractere

- semnifică o secvență lexicografică

Pe baza acestor metacaractere, `shell`-ul va genera nume de fișiere utilizate în cadrul comenzilor.

Exemple:

*.c - indică toate fișierele având sufixul c;

Def[0-9] - indică fișierele def0, def1, ..., def9;

Prog[09] - indică fișierele prog0, prog9;

cap? - indică fișierele existente cap1, cap2, cap3;

cap* - indică fișierele existente cap1, cap2, cap3, cap1.c, cap2.c;

prog[*?] - indică atât varianta prog*, cât și varianta prog?.

p*r - face referire la cele care încep cu p și se termină cu r;

c - face referire la cele care conțin litera c.

[abc]* - face referire la toate fișierele al căror nume începe cu a,b,c;

*[[I-N1-3] - face referire la fișierele ce se termină cu I,J,K,L,M,N,1,2,3;

Observații:

1. Cele trei construcții lexicale comanda, [indicatori], [argument1] [argument2] .., trebuie să fie separate prin spații.

2. Toate comenzile acceptă intrări de la intrarea standard (de regulă, tastatura), afișează ieșirea la ieșirea standard (de regulă, ecranul terminalului).

3. UNIX oferă posibilitatea introducerii în linia de comandă a mai multor comenzi ce se vor executa serial. Comenzile trebuiesc separate prin caracterul ;.

3.4.2. Gestiunea utilizatorilor și grupurilor

UNIX este un sistem multiuser și multisesiune, utilizatorii putând avea deschise mai multe sesiuni de lucru pe un același calculator sau pe calculatoare diferite din rețea.

Gestiunea utilizatorilor în cauză, precum și a grupurilor din care aceștia fac parte, reprezintă o sarcină importantă a administratorului de sistem. Pentru îndeplinirea acesteia UNIX-ul pune la dispoziție câteva instrumente și convenții care fac ca această sarcină să fie mai ușor de realizat. Astfel, accesul la resursele unui sistem UNIX se realizează, de regulă, prin intermediul unor conturi utilizator ce sunt setate de către administratorul de sistem, ulterior instalării sistemului de operare. Accesul se poate realiza și prin intermediul unor conturi sistem, dintre acestea cel mai important fiind cel aferent utilizatorului `root` (superuser).

Prin folosirea de conturi separate pentru fiecare utilizator, securitatea sistemului este mult mai bine asigurată, corespondența de unu-la-unu între utilizatori și conturi făcând ca activitatea de gestionare a acestora să fie mai ușoară.

Contul utilizatorului `root`

Este un cont creat automat la momentul instalării sistemului de operare UNIX. În timp ce majoritatea conturilor utilizatorilor sunt setate astfel încât să se prevină distrugerea accidentală a sistemelor de fișiere, contul `root` nu prezintă nici o restricție, accesul său la toate resursele sistemului, fiind deplin. Din acest motiv, folosirea acestui cont revine numai administratorului de sistem, fiind utilizat în realizarea operațiilor de configurare și întreținere a sistemului UNIX.

Pentru prevenirea apariției unor incidente nedorite datorate folosirii necorespunzătoare a acestui cont, este indicată crearea unor conturi asociate unor nume de utilizator, care să aibă ca scop realizarea anumitor sarcini de administrare a sistemului, sarcini ce nu necesită un acces deplin la resursele acestuia. De exemplu, se poate seta un cont utilizator destinat efectuării copiilor de siguranță (`backup`), un altul pentru accesul la poșta electronică, la Internet, etc.

Protecția contului utilizatorului `root`, dar și a celorlalte conturi utilizator, se realizează prin asignarea de parole. Alegerea acestora este indicat să se facă astfel încât depistarea lor de către ceilalți utilizatori ai sistemului, să fie dificilă, iar schimbarea lor periodică se constituie într-o măsură de securitate obligatorie.

Nume de utilizator implicite (standard)

Așa cum am menționat anterior, există o serie de nume de utilizator implicite ale căror conturi sunt create în timpul procesului de instalare a sistemului de operare (în fapt este vorba de crearea fișierului `/etc/passwd`).

Aceste nume sunt utilizate de către sistemul de operare și administratorul de sistem, pentru realizarea unor scopuri speciale.

Fișierul `/etc/passwd`

Toate informațiile referitoare la conturile utilizatorilor sunt păstrate în fișierul `/etc/passwd`. Acest fișier se află în proprietatea utilizatorului `root` și are identificatorul grupului (GID) setat la zero. Permișiunea de scriere este setată numai utilizatorului `root`, ceilalți utilizatori având numai permișiunea de citire.

Liniile din fișierul `/etc/passwd` sunt divizate în următorul format strict:

```
nume_utilizator:parola:UID:GID:comentariu:director_implicit:comanda_login
```

Fiecare linie este alcătuită din câmpuri ce sunt separate prin simbolul `:`. Dacă într-un câmp nu este nimic introdus, acesta va rămâne gol, iar simbolul `:` se va păstra pentru a asigura că fiecare linie conține șapte câmpuri. Denumirea câmpurilor (de la stânga la dreapta pentru fiecare linie) și semnificația lor, este următoarea:

Câmp	Semnificație
<code>nume_utilizator</code>	Un identificator unic pentru utilizator
<code>parola</code>	Parola utilizatorului (criptată)
<code>UID</code>	Un număr unic care identifică utilizatorul pentru sistemul de operare
<code>GID</code>	Un număr unic care identifică grupul la care aparține utilizatorul
<code>comentariu</code>	Uzual, numele real al utilizatorului, sau o altă informație referitoare la acesta
<code>director_implicit</code>	Directorul în care utilizatorii sunt plasați când se conectează la sistem (director home)
<code>comanda_login</code>	Comanda executată când utilizatorul se conectează, în mod normal un shell

Descrierea detaliată a câmpurilor respective este prezentată în continuare.

`nume_utilizator`

Este un câmp ce conține un șir format din opt caractere sau mai puțin, ce identifică în mod unic fiecare utilizator. În specificarea numelui se pot folosi, pe lângă litere, și linia de subliniere, numerele, virgula și anumite caractere speciale. Deoarece cele mai multe comenzi UNIX sunt scrise cu litere mici, convenția este ca numele de utilizator să fie și el scris tot cu litere mici.

`parola`

În acest câmp sistemul stochează parola criptată a utilizatorului. Informația respectivă este foarte sensibilă la modificări necorespunzătoare, acestea putând conduce la blocarea contului utilizator căruia îi este asociată. Parola poate fi modificată numai de către administratorul de sistem (care s-a conectat cu numele de utilizator `root`), sau de către utilizatorul însuși. Comanda folosită în acest scop este `passwd`.

Observație:

Anumite versiuni de UNIX, datorită unor probleme potențiale de securitate, nu stochează parolele utilizatorilor în fișierul `/etc/passwd`.

La momentul conectării unui utilizator la sistem, parola introdusă de acesta se compară în mod logic cu un bloc de zero-uri, iar rezultatul este comparat cu intrarea corespunzătoare din fișierul de parole. Utilizatorului îi este permis accesul numai dacă cele două informații se potrivesc.

Câmpul de parolă este folosit pentru restricționarea accesului la sistem. Astfel, dacă se dorește ca un cont să nu poată fi folosit pentru accesarea sistemului, se plasează în câmpul de parolă corespunzător lui, un asterisc. În exemplul prezentat anterior, se poate observa că multe conturi de utilizator standard au un asterisc în câmpul de parolă, lucru ce efectiv blochează accesul.

În situația în care acest câmp nu conține nimic (este gol), la contul respectiv se permite un acces nerestricțiv. Prin urmare, oricine poate folosi numele de utilizator respectiv pentru a i se acorda accesul imediat, fără a i se solicita introducerea unei parole.

Observație:

Nu se va completa câmpul de parolă cu o parolă (ed în d use fișierul `/etc/passwd`) deoarece în acest fel, nu este posibilă recrearea parolei criptate, iar contul utilizatorului respectiv se va bloca.

UID

Fiecare nume de utilizator are asociat un identificator unic (UID). Acesta este folosit de către sistemul de operare UNIX pentru a identifica anumite informații ce sunt asociate cu utilizatorul. Folosirea UID în locul numelui utilizatorului este preferabilă, deoarece cu numerele se lucrează mai ușor decât cu caracterele, și în plus ocupă și mai puțin spațiu¹¹.

Numerele UID sunt în general asignate într-un domeniu specific. Cele mai multe sisteme UNIX, de exemplu, alocă numerele de la 0 la 99 pentru conturile utilizator standard, și numere UID de la 100 în sus pentru ceilalți utilizatori. Asignarea numerelor UID pentru utilizatorii obișnuiți ai sistemului, este bine să se facă secvențial.

GID

Identificatorul grupului, GID, este un număr folosit pentru a păstra urma grupului la care utilizatorii aparțin când ei se conectează la sistem (grup de `startup`). Numerele GID se

¹¹De exemplu, sistemul de operare Linux păstrează urma tuturor proceselor lansate în execuție de către utilizator, folosind UID și nu numele acestuia.

situează într-un domeniu ce începe cu 0 ș.am.d., numărul GID 100 fiind asignat grupului `users` ce conține toți utilizatorii neprivilegiați ai sistemului.

comentariu

Acest câmp este folosit de către administratorul de sistem pentru adăugarea oricărei informații pe care acesta o consideră necesară în identificarea utilizatorului. Tipic, această zonă este folosită pentru a introduce numele complet al utilizatorului, cu toate că anumiți administratori de sisteme preferă să adauge alte informații (de exemplu numele departamentului unde lucrează utilizatorul, sau numărul de telefon al acestuia).

Anumite comenzi ale sistemului de operare pot să folosească acest câmp pentru a afișa diverse informații despre utilizatori, prin urmare conținutul său poate fi alterat.

directorul_implicit (home directory)

Acest câmp specifică directorul în care va fi comutat utilizatorul după ce acesta s-a conectat la sistem (directorul curent). Fiecare utilizator din sistem trebuie să aibă dedicat un director implicit propriu¹², variabila de mediu `HOME` din cadrul fișierelor de `startup` fiind inițializată cu această valoare. Directoarele implicite ale utilizatorilor sunt localizate într-un director comun, numit `/home`, ce este creat la momentul instalării sistemului `Linux`. Calea aferentă directoarele implicite găsite aici este de forma: `/home/nume_utilizator`.

Alte versiuni de `UNIX` folosesc ca directoare implicite directoarele `/usr` sau `/u`.

comanda_login

Reprezintă comanda ce va fi executată când procedura de conectare se termină. În cele mai multe cazuri se lansează în execuție o comandă `shell`, cum este `cs`h sau `bs`h, pentru a furniza utilizatorului un interpretor de comenzi (`C Shell`, respectiv `Bourne Shell`).

Dacă câmpul este lăsat gol, sistemul de operare lansează ca implicit `shell`-ul `Bourne`. Schimbarea `shell`-ului ce va fi lansat poate fi realizată cu ajutorul comenzilor `chsh` sau `passwd -s`. Indiferent de comanda ce este folosită, verificarea faptului că aceasta este permisă se face prin căutarea ei în fișierul `/etc/shells`. Numai comenzile care se află în acest fișier sunt permise ca și intrări valide. Dacă sistemul folosește fișierul `/etc/shells`, trebuie verificat ca permisiunile de acces asupra sa și proprietarul să fie aceleași ca ale fișierului `/etc/passwd`, altfel un utilizator își va putea modifica `comanda_login`.

Observații:

1. În cazul în care se dorește invalidarea temporară a unui cont utilizator se va plasa un asterisc ca prim caracter al parolei criptate. Nu se va altera nici un caracter al parolei existente, ci numai se va adăuga un asterisc în fața acesteia. Când se dorește reactivarea contului se va elimina asteriscul introdus anterior.

2. Din cauza unor potențiale probleme de securitate, anumite versiuni de `UNIX` nu păstrează parolele în fișierul `/etc/passwd`. Dacă în câmpul de parolă din cadrul tuturor liniilor ce compun acest fișier, întâlnim caracterul `x`, atunci pentru stocarea parolelor este folosit un alt fișier, numit `shadow password file`, situat în directorul `/etc/shadow`, și care poate fi citit numai de către utilizatorul `root`.

Grupuri

Fiecare utilizator al unui sistem `UNIX` aparține unui grup. Un grup este o colecție de utilizatori realizată în virtutea faptului că membrii ei au o caracteristică comună (de exemplu, pot să lucreze toți în cadrul aceluiași departament, pot avea acces la un set de programe particular, pot avea acces la un dispozitiv special cum este un scanner sau o imprimantă laser, etc.) În

¹²În cadrul sistemului `Linux`, acesta poartă chiar numele utilizatorului.

general utilizatorii ce aparțin unui grup, au aceleași permisiuni de acces asupra unor fișiere și directoare particulare.

Utilizatorii pot aparține mai multor grupuri, dar la un moment dat, un utilizator poate fi membru numai al unui grup (grup primar). Aceasta deoarece la orice moment de timp, este permis un GID per utilizator.

Datorită faptului că grupurile pot avea setul lor de permisiuni de acces asupra unor fișiere și directoare, apartenența sau nu a unui utilizator la un grup anume, poate determina sau restricționa accesul la fișierele și directoarele respective.

Grupuri implicite (standard)

În urma procesului de instalare a sistemului de operare UNIX sunt create mai multe grupuri implicite (standard) (în fapt este vorba de crearea fișierului `/etc/group`).

Apartenența la aceste grupuri este restricționată pentru utilizatorii obișnuiți deoarece aceasta implică obținerea unor permisiuni de acces care sunt aceleași cu cele ale utilizatorului `root`. Ca urmare membri acestor grupuri sunt reprezentați de utilizatorii implicați ai sistemului.

Informația referitoare la grupuri este stocată în fișierul `/etc/group`, care este similar în format cu fișierul `/etc/passwd`.

Fiecare grup are o linie proprie în cadrul acestui fișier, iar fiecare linie este compusă din patru câmpuri ce sunt separate prin simbolul `:`. Două simboluri `:` imediat alăturate semnifică faptul că nu a fost specificată o valoare anume pentru câmpul respectiv (câmpul este gol).

Formatul aferent fiecărei linii din cadrul fișierului `/etc/group` este următorul:
`nume_grup:parola_grup:GID:utilizatori`

unde:

`nume_grup`

Este un câmp ce conține un nume unic alcătuit, în general, din opt caractere sau mai puțin (se folosesc de regulă numai caractere alfanumerice) și care specifică numele grupului respectiv.

`parola`

Acest câmp, de regulă, fie este liber, fie conține un asterisc. El poate conține totuși și o parolă, pe care utilizatorul trebuie să o introducă dacă dorește să se asocieze grupului. Deși nu toate versiunile de UNIX folosesc câmpul în cauză, el este păstrat din rațiuni de compatibilitate cu versiunile mai vechi.

`GID`

Ca și în cazul fișierului `/etc/passwd`, acest câmp este folosit pentru a stoca un număr ce reprezintă identificatorul grupului.

`utilizatori`

Este un câmp ce conține o listă a tuturor utilizatorilor ce aparțin aceluși grup.

3.4.3. Permisuniile de acces asupra fișierelor și directoarelor

Sistemul de operare UNIX este un sistem multiuser, iar permisuniile de acces asupra fișierelor și directoarelor reprezintă una dintre facilitățile de securitate ale acestui sistem.

În fapt, este vorba de o modalitate prin care sistemul protejează fișierele și directoarele împotriva oricărui acces neautorizat, intenționat sau accidental.

Toate fișierele și directoarele UNIX au permisiuni de acces și proprietari.

La momentul creării unui fișier sau director, utilizatorul ce le-a creat devine în mod automat proprietarul acestora (`owner`). Aceasta înseamnă că el are privilegiul de a schimba permisuniile (specifică cine și ce permisiuni are) sau poate schimba proprietarul fișierului

(directorului)¹³.

Schimbarea permisiunilor de acces sau a proprietarului are drept scop furnizarea unui acces mai complet asupra fișierelor și directoarelor.

Schimbarea proprietarului unui fișier sau director se face cu comanda:

```
chown proprietar_nou nume_fisier
```

Utilizatorii (dar și fișierele și directoarele) aparțin grupurilor. Grupurile reprezintă o modalitate convenabilă de a furniza autorizări de acces mai multor utilizatori, dar nu oricărui utilizator din sistem. La momentul creării unui utilizator, grupul implicit al acestuia este cel ce îi poartă numele și care este creat odată cu utilizatorul.

Comanda prin care se poate schimba grupul căruia îi aparține un fișier (director) este:

```
chgrp grup_nou nume_fisier
```

Proprietarul fișierului trebuie să fie membru al noului grup (`grup_nou`).

Observație:

Grupul `users` este cel ce conține toți utilizatorii neprivilegiați ai sistemului.

Permisiunile de acces asupra fișierelor (directoarelor) acceptate de UNIX sunt:

- *permisiunea de citire* (`r`) - permite vizualizarea (citirea) conținutului unui fișier. În cazul unui director permite afișarea conținutului acestuia (utilizând, de exemplu, comanda `ls`).
- *permisiunea de scriere* (`w`) - permite modificarea, ștergerea și redenumirea unui fișier. În cazul unui director existența acestei permisiuni permite actualizarea, ștergerea și redenumirea directorului.
- *permisiunea de execuție* (`x`) - permite executarea fișierului prin tastarea numelui acestuia. Pentru directoare permite accesul (comutarea) în directorul respectiv, și efectuarea de operații asupra fișierelor din cadrul său.

Observații:

1. Dacă un utilizator are permisiunea de scriere asupra unui director ce conține fișiere, permisiunile asupra fișierelor respectiv sunt rescrise de permisiunile pe acel director.

2. Orice utilizator care are permisiunea de a citi un fișier poate să copieze acel fișier. Când un fișier este copiat, copia se află în proprietatea utilizatorului care a efectuat copierea. Acesta poate să schimbe proprietarul și permisiunile, să editeze fișierul ș.a.m.d.

3. Ștergerea permisiunii de scriere asupra unui fișier nu permite ca acesta să fie șters.

Pentru a vizualiza permisiunile existente asupra unui fișier (director) se folosește comanda:

```
ls -l nume_fisier
```

De exemplu, se tastează:

```
ls -l test.txt
```

Se va afișa:

```
-rw-rw-r--l student student 150 Dec 19 08:08 test.txt
```

Interpretarea informației afișate se face astfel:

Primul caracter din șirul afișat indică tipul fișierului. Astfel, dacă acest caracter este `-` atunci este vorba de un fișier normal. Dacă caracterul este `d`, atunci este vorba de un director, iar dacă caracterul este `l` este vorba de o legătură simbolică către un alt program sau fișier din sistem¹⁴.

Următoarele nouă caractere din șirul afișat specifică, în seturi de câte trei, permisiunile de

¹³Utilizatorul root are permisiunea de a citi, scrie, și executa orice fișier din sistem, indiferent cine este proprietarul acestora.

¹⁴Sunt posibile și alte caractere.

acces pentru următoarele trei categorii diferite de entități:

- proprietarul (`owner`) fișierului (directorului);
- grupul (`group`) (utilizatorii ce aparțin aceluiași grup ca și proprietarul fișierului /directorului)¹⁵;
- alții (`others`) (utilizatorii și grupurile altele decât proprietarul și cei din grupul căruia îi aparține proprietarul).

Dacă unul din caracterele ce specifică permisiunile (`r`, `w`, `x`) va apărea ca fiind înlocuit prin caracterul `-`, atunci permisiunea respectivă, pentru entitatea în cauză, nu va fi acordată.

Alte informații afișate de către comanda `ls -l` includ: numele fișierului, data și timpul creării sale, mărimea acestuia.

Schimbarea permisiunilor de acces asupra fișierelor

Pentru a schimba permisiunile de acces asupra fișierelor se utilizează comanda:

```
chmod indicator nume_fisier.
```

unde `indicator` se va înlocui cu identitatea entității pentru care se face schimbarea de permisiune:

- `u` utilizatorul care este proprietar al fișierului (`owner`)
- `g` grupul căruia îi aparține utilizatorul
- `o` alții (alții decât utilizatorul și grupul acestuia) (`others`)
- `a` oricine (`u`, `g` și `o`) (`everyone`)

urmată de tipul acțiunii ce se dorește:

- `+` adaugă o permisiune
- `-` elimină o permisiune
- `=` marchează permisiunea ca fiind singura acceptată

și litera (literele) aferente permisiunilor (`r`, `w`, `x`).

Observații:

1. În cadrul sintaxei comenzii `chmod`, `indicator` se poate înlocui și cu un număr format din trei cifre, a cărui codificare este explicată în continuare.

2. Atât timp cât suntem proprietar al unui fișier (director), sau suntem conectați ca utilizator privilegiat (`root`), putem să modificăm permisiunile fișierului (directorului) în orice combinație pentru proprietar, grup și alții.

Există două modalități de modificare a permisiunilor.

O primă modalitate este cea în care se folosesc literele. Pentru a specifica entitățile pentru care se schimbă permisiunile, se vor tasta literele `u`, `g`, `o` sau `a` în sintaxa comenzii `chmod`. Litera (literele) vor fi urmate de unul dintre semnele `+`, `-`, sau `=`, pentru a adăuga, șterge sau marca permisiunea ca fiind singura acceptată. Semnul este urmat de litera permisiunii respective (`r`, `w`, `x`).

Exemple:

Presupunem că permisiunile inițiale ale fișierului `test.txt` sunt:

```
-rw-rw-r--1 student student 150 Dec 19 08:08 test.txt
```

Conform permisiunilor afișate proprietarul (`student`) și grupul (`student`) pot citi și scrie în fișier. Oricine alt utilizator ce nu aparține grupului `student` poate numai să citească fișierul.

Dacă tastăm comanda:

```
chmod o+rw test.txt
```

numai alții (ce nu sunt nici proprietari și nici nu aparțin grupului `student`) vor putea să citească și să scrie fișierul.

¹⁵Grupul din care face parte proprietarul.

Deoarece utilizatorul student este proprietar al acestui fișier, el va putea întotdeauna să schimbe permisiunile pentru a recăpăta accesul în citire și scriere.

Comanda ce se va tasta va fi:

```
chmod u+rw test.txt
cat test.txt
acesta este un test
```

S-a verificat și permisiunea în citire cu ajutorul comenzii `cat`.

Pentru ștergerea tuturor permisiunilor asupra fișierului `test.txt` pentru oricine, se va tasta comanda:

```
chmod a-rw test.txt
```

Să vedem dacă fișierul poate fi citit:

```
cat test.txt
cat: test.txt: Permission denied
```

Mesajul de eroare returnat de sistem, `Permission denied`, semnifică faptul că execuția comenzii anterioare (`cat`), nu este posibilă datorită lipsei permisiunii de citire necesare.

Alte exemple de setări ce pot fi folosite în cadrul comenzii `chmod` sunt:

```
g+w   adaugă permisiunea de scriere pentru grup
o-rwx șterge toate permisiunile pentru alții
u+x   permite proprietarului fișierului să execute fișierul
a+rw  pemite oricui să citească și să scrie în fișier
ug+r  permite proprietarului și grupului să citească fișierul
g=rx  permite grupului numai să citească și să execute fișierul, fără scriere
```

O a doua metodă de modificare a permisiunilor asupra fișierelor este cea bazată pe un sistem de codificare numerică. Acesta permite specificarea permisiunilor asupra fișierelor sub forma unui număr din 3 cifre octale.

Este important de a înțelege cum lucrează acest sistemul de codificare, deoarece numerele în cauză sunt folosite, atât pentru a schimba permisiunile asupra fișierelor, cât și de către mesajele de eroare ce implică permisiunile.

În cadrul unui astfel de număr prima cifră codifică permisiunile pentru proprietar, a doua pe cele pentru grup și a treia pe cele pentru alții.

Conform sistemului de codificare respectiv, seturile de câte trei permisiuni (`rxw`) pot fi interpretate ca un număr binar pe 3 biți. Astfel, o permisiune acordată corespunde unei cifre de 1, iar o permisiune neacordată corespunde unei cifre de 0. Prin urmare pentru setul (`r-x`) combinația binară corespondentă va fi `101` care în zecimal este egală cu: $4+0+1=5$.

Cifrele individuale sunt codificate prin însumarea tuturor permisiunilor, "permise" pentru acel utilizator particular¹⁶ după cum urmează:

Permisiune citire	4
Permisiune scriere	2
Permisiune execuție	1

Sunt posibile următoarele combinații:

- 0 - nici o permisiune
- 4 - numai citire
- 2 - numai scriere
- 1 - execuție
- 6 - citire și scriere
- 5 - citire și execuție
- 3 - scriere și execuție

¹⁶Deci dintr-un set de trei.

7 - toate

Prin urmare o permisiune asupra unui fișier, codificată prin numărul 751, înseamnă că proprietarul are permisiunile r, w, x ($4+2+1=7$), grupul are permisiunile r și x ($4+0+1=5$) și alții au numai permisiunea x ($0+0+1=1$).

Observație:

Setarea permisiunilor 666 sau 777 va permite ca orice utilizator să poată citi și scrie într-un fișier sau director. Asemenea setări pot determina fraudarea fișierelor, și ca urmare nu sunt recomandate.

În continuare sunt prezentate câteva dintre cele mai obișnuite setări, valori numerice, precum și semnificația lor:

-rw----- (600) - numai proprietarul are permisiunile r și w
-rw-r--r-- (644) - numai proprietarul are permisiunile r și w ; grupul și alții pot numai să citească
-rwx----- (700) - numai proprietarul are permisiunile $r w x$
-rwxr-xr-x (755) - proprietarul are permisiunile $r w$ și x ; grupul și alții pot numai să citească și să execute
-rwx--x--x (711) - proprietarul are permisiunile $r w x$; grupul și alții pot numai să execute
-rw-rw-rw- (666) - oricine poate să citească și să scrie în fișier
-rwxrwxrwx (777) - oricine poate să citească, să scrie, să execute.

Schimbarea permisiunilor asupra directoarelor

Se poate realiza exact în același fel ca și în cazul fișierelor, deci tot cu ajutorul comenzii `chmod`. Orice utilizator care are permisiunea de scriere într-un director poate șterge fișiere din acel director, chiar dacă utilizatorul are sau nu permisiunea de scriere asupra fișierului.

Deoarece nu putem "executa" un director, când se setează sau șterge permisiunea de execuție asupra acestuia, în fapt se setează sau se șterge permisiunea de a căuta în acel director.

De exemplu să tastăm:

```
chmod a-x aplicatii
```

pentru a șterge permisiunea de execuție pentru toți utilizatorii.

Iată ce se întâmplă când se încearcă folosirea comenzii `cd` în directorul `aplicatii`:

```
cd aplicatii
```

```
bash:aplicatii: Permission denied
```

Să restaurăm permisiunea de execuție pentru proprietar și pentru grupul acestuia:

```
chmod ug+x aplicatii
```

Acum, dacă vom verifica cu ajutorul comenzii `ls -dl` se va vedea că numai `others` au interzis accesul la directorul `aplicatii`.

Pentru a permite oricui de a avea acces în citire și scriere asupra directorului `aplicatii` se va tasta:

```
chmod -R a+rw aplicatii
```

Prin adăugarea indicatorului `-R`, se pot schimba permisiunile pentru întreaga structură director.

Observații:

1. Modalitatea de a obține acces la un fișier atunci când nu avem permisiunile necesare, și fără a ne deconecta de la sistem, este de a utiliza comanda `su`. Acesta permite conectarea cu numele altui utilizator ce are permisiunile necesare. Evident va trebui să cunoaștem parola utilizatorului respectiv.

2. Unele versiuni de UNIX furnizează un bit suplimentar numit `sticky bit` ca parte a permisiunii asupra unui director. Scopul acestui bit este de a permite numai proprietarului

directorului, proprietarului fișierului sau utilizatorului `root` să șteargă și să redenumescă fișiere.

3.4.4. Sistemul de fișiere al UNIX

Un sistem de fișiere reprezintă modul de organizare și exploatare a informațiilor stocate pe un suport de memorie externă în vederea accesării și prelucrării lor de către sistemul de operare.

Din punctul de vedere al utilizatorului, sistemele de fișiere prezintă o organizare bazată pe conceptele de fișier și director.

Fișierele sunt entități care încapsulează informația de un anumit tip, iar directoarele grupează în interiorul lor fișiere și alte directoare.

Deci sistemul de fișiere are o organizare ierarhică, fișierele fiind grupate în directoare care sunt structurate pe mai multe niveluri într-o structură arborescentă. Directorul de nivel cel mai înalt, din cadrul acestei arborescențe, se numește *director rădăcină*, și este simbolizat prin caracterul `/`.

Orice fișier sau director poate fi identificat prin numele său indicat ca nume de cale, fie în mod absolut, față de directorul rădăcină, fie în mod relativ, față de directorul curent.

3.4.4.1 Partiții

Un sistem de fișiere este în general creat pe hard disc. Sub sistemul de operare UNIX este posibilă instalarea mai multor sisteme de fișiere, fiecare dintre acestea fiind plasat într-o zonă proprie pe hard disc. Aceste zone în care este subdivizat hard discul se numesc partiții¹⁷.

La nivel utilizator, fiecare partiție se comportă ca un disc de sine stătător memorând un sistem de fișiere. Informația referitoare la partiții se memorează la începutul discului, în așa-numita *tabelă de partiții*. Aceasta conține 4 intrări (câte una pentru fiecare partiție posibilă) în care sunt memorate pozițiile, dimensiunile și tipurile partițiilor de pe disc. Cele 4 partiții se numesc partiții primare, fiind posibil ca în interiorul oricăreia dintre ele să se creeze câte o nouă tabelă de partiții, referind partiții care fizic se află în interiorul partiției curente și care se numesc partiții extinse.

Sistemul de operare UNIX necesită prezența a cel puțin o partiție. Partiția necesară menționată anterior se numește partiție `root`. În cadrul ei rezidă software-ul de sistem și fișierele de configurare. Deoarece partiția `root` nu are menirea de a păstra datele utilizatorilor, este necesară crearea de partiții separate pentru directoarele `home` ale utilizatorilor, pentru fișierele temporare, ș.a.

În UNIX, echipamentelor periferice le sunt asociate fișiere de dispozitiv, ceea ce permite utilizarea și în cazul acestora, a comenzilor ce sunt folosite pentru fișierele obișnuite. Acest lucru este valabil și pentru hard discuri. Ca urmare, pentru identificarea lor se folosește următoarea schemă, ce este funcție de tipul unității de control la care s-a conectat hard discul.

Astfel, sub sistemul `Linux`, hard discurile conectat la o unitate de control `IDE`, sunt denumite:

```
/dev/hd[unitate_hard_disc]
```

Fiecare unitate de hard disc `IDE` este identificată printr-o literă. Astfel, discul `master` (primul) din cadrul primei unități de control este `a`, cel de-al doilea disc (`slave`) din cadrul aceleiași unități de control este `b`. Primul disc atașat celei de a doua unități de control este `c`, ș.a.m.d.

Pentru a referi partițiile create pe astfel de hard discuri, se utilizează numere de ordine. De exemplu, cea de a treia partiție a unității de hard disc `slave` de pe prima unitatea de

¹⁷Fiecare partiție este o mulțime continuă de blocuri. Un bloc este format dintr-un sigur sector (cum este cazul la dischete) sau din mai multe sectoare (cazul hard discurilor).

control este referită: /dev/hdb3.

Unitățile de hard disc conectate unei unități de control SCSI folosesc aceeași schemă de identificare, cu excepția că în loc să utilizeze drept prefix construcția /dev/hd, folosesc /dev/sd, urmată de litera corespunzătoare. Referirea partițiilor este similară.

Astfel, când ne referim la a doua partiție de pe primul hard disc atașat unei unități de control SCSI, vom folosi specificația: /dev/sda2. Atunci când dorim să ne referim la întregul disc, vom specifica toată informația cu excepția numărului asociat partiției. De exemplu, pentru a ne referi la întregul disc master conectat pe prima unitatea de control SCSI, vom utiliza specificația: /dev/sda.

În cele două scheme de identificare anterioare, /dev este numele directorului aparținând arborescenței standard UNIX, ce conține fișierele de dispozitiv asociate echipamentelor periferice din cadrul sistemului.

3.4.4.2. Tipuri de fișiere

În cadrul sistemului de fișiere al UNIX există mai multe tipuri de fișiere. Acestea sunt:

- fișiere normale (obișnuite)
- directoare (fișiere catalog)
- legături hard
- legături simbolice
- socket-uri
- conducte cu nume (fișiere FIFO)
- fișiere de dispozitiv (fișiere speciale)

Fișiere normale

Un fișier normal este un fișier permanent, ce este privit de către sistemul de operare ca un șir de octeți fără o organizare logică specială, și a cărui structură internă este irelevantă din punctul de vedere al administratorului de sistem.

Un astfel de fișier poate conține informație binară (în cazul unui fișier în format executabil), sau linii de text separate prin caracterul linie nouă (012₈), structurarea sa logică revenind exclusiv în sarcina programatorului.

O comandă `ls -l` va afișa în cazul unui fișier normal, o informație similară cu următoarea:

```
-rw----- 1 student ise 42 May 12 13:09 fistest
```

Se observă că primul caracter afișat este `-`. Acesta semnifică faptul că fișierul respectiv este un fișier normal.

Directoare

Sunt fișiere de un tip special ce conțin o listă de alte fișiere. Informațiile din lista respectivă fac referire la numele și locațiile fișierelor, mărimea lor, timpul aferent creării și modificării acestora.

Directoarele pot ele înșile să conțină subdirectoare, care la rândul lor pot să conțină alte subdirectoare, ș.a.m.d., formând în acest fel o structură ierarhică (arborescentă).

Se poate aprecia deci că fișierele director reprezintă o modalitate de a structura logic sistemul de fișiere.

Un director poate fi citit numai de către sistemul de operare UNIX sau de către programe special scrise pentru a efectua procesarea acestuia.

Cu ajutorul comenzii `ls -l` se pot identifica directoarele datorită apariției caracterului `d` ce este plasat înaintea permisiunilor de acces:

```
drwx----- 2 student ise 512 May 12 13:08 public
```

3.4.4.3 Legături la fișiere

În anumite circumstanțe este necesar¹⁸, de a furniza nume alternative pentru un același fișier.

Acest lucru poate fi realizat prin legarea unui nume de fișier la altul, o legătură fiind o modalitate de a furniza un alt nume unui același fișier¹⁹. Legarea unui fișier la un alt nume nu implică duplicarea conținutului fișierului respectiv.

Este posibil de a lega un fișier la un alt nume în cadrul acelui director sau la același nume în cadrul unui alt director.

Orice operație care se execută asupra fișierului legătură (mai puțin ștergerea) își va avea efectul de fapt asupra fișierului indicat de legătură. Dacă este solicitată ștergerea, efectul depinde de tipul legăturii respective.

Legături hard (hard links)

Acest tip de legătură creează o referință (un pointer) către un fișier deja existent, fără duplicarea conținutului fișierului respectiv. Deci, numele original al fișierului și numele fișierului legătură fac referire către aceeași adresă fizică (același i-nod).

Ca urmare, la afișarea conținutului unui director acesta apare că ar conține două fișiere identice, lucru ce determină sistemul să le trateze în consecință.

Există două limitări importante ale unei legături hard, și anume: un director nu poate avea o legătură hard, și o astfel de legătură nu poate exista peste mai multe sisteme de fișiere deoarece ea partajează un i-nod.

Este posibil de a șterge numele fișierului original fără a șterge numele fișierului legătură. Sub aceste circumstanțe, fișierul nu este șters, dar intrarea în director a fișierului original va fi ștearsă iar contorul numărului de legături este decrementat cu 1. Blocurile de date ale fișierului original sunt șterse atunci când contorul numărului de legături devine zero.

Legături simbolice (symbolic links)

Legăturile simbolice sunt de fapt fișiere distincte, marcate cu un cod special, care au ca și conținut numele complet al fișierului indicat.

Există deci două fișiere: unul este fișierul original, și altul este fișierul legătură ce conține numele fișierului original.

Spre deosebire de o legătură hard, o legătură simbolică poate să existe peste mai multe sisteme de fișiere și poate fi folosită pentru a lega atât directoare cât și fișiere.

Dezavantajul unui legăturii simbolice este că pentru ea (fiind fișier) trebuie creat un i-nod separat și, în plus, ocupă spațiu pe disc prin conținutul ei.

Ștergerea unei astfel de legături nu afectează fișierul original, dar dacă acesta este șters, legătura simbolică va face referire către un fișier care nu mai există.

Pentru a genera legături hard sau simbolice la fișiere, se utilizează o comandă specifică, numită `ln`. Sintaxa ei generală este:

```
ln nume_fisier nume_legatura
```

unde `nume_fisier` este fișierul ce există deja, iar `nume_legatura`, este numele fișierului legătură creat.

Implicit comanda creează o legătură hard.

De exemplu, dacă în directorul curent există un fișier numit `test1`, și tastăm comanda `ls -l`, se vor afișa următoarele informații:

```
-rw----- 1 stud1 ise 42 May 12 13:04 test1
```

Pentru a lega hard fișierul `test1` la fișierul `test2` în directorul curent, vom executa comanda:

```
ln test1 test2
```

¹⁸De exemplu, dacă dorim să permitem accesul altor utilizatori, la propriile noastre fișiere, fără ca pentru aceasta să le punem la dispoziție o copie a fișierelor respective.

¹⁹Practic, o legătură este văzută de către utilizator ca un fișier cu un nume propriu, dar care în realitate face referire la un alt fișier de pe disc.

Tastând din nou comanda `ls -l`, se vor afișa informațiile:

```
-rw----- 2 stud1 ise 42 May 12 13:04 test2
-rw----- 2 stud1 ise 42 May 12 13:04 test1
```

Apar deci două fișiere care au aceeași mărime, 42. Totodată numărul de legături (coloana a doua), a fost incrementat de la 1 la 2.

Tastând comanda `ls -il`, se vor afișa informațiile:

```
9180 -rw----- 2 stud1 ise 42 May 2 8:04 test2
9180 -rw----- 2 stud1 ise 42 May 2 8:04 test1
```

Se observă că ambele nume de fișier fac referire către un același `i-nod`, 9180. Este deci vorba de două nume care fac referire către un același fișier.

Unul dintre indicatorii cei mai utilizați ai comenzii `ln` este `-s`, folosirea acestuia permițând generarea unei legături simbolice. Formatul comenzii `ln`, în acest caz, este:

```
ln -s nume_fisier nume_legatura
```

De exemplu, pentru a crea o legătură simbolică la fișierul `test1` în directorul curent, se va tasta comanda următoare:

```
ln -s test1 test2
```

Aceasta creează un fișier `test2` legat, care va conține numele lui `test1`.

Tastând o comandă `ls -l` în directorul curent, se vor afișa informațiile:

```
-rw----- 2 stud1 ise 42 May 12 13:04 test1
lrw----- 1 stud1 ise 42 May 12 13:04 test2->test1
```

Fișierul `test2` apare marcat ca fiind o legătură simbolică în cadrul sistemului de fișiere.

Socket-uri

Un `socket` nu este în fapt un fișier real, ci un mecanism folosit de UNIX pentru a comunica între două calculatoare gazdă, utilizând pentru aceasta porturi rețea.

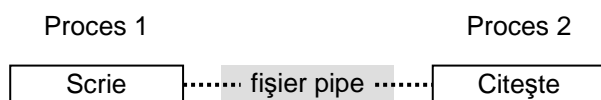
Fișierele `socket` sunt identificate de către setările lor de permisiuni ce încep cu caracterul `s`. O comandă `ls -l` asupra unui fișier `socket` afișează o informație similară cu următoarea:

```
srwxrwxrwx 1 root admin 0 May 10 14:38 X0
```

Pentru ștergerea unui fișier `socket` se va folosi comanda `rm`.

Conducte cu nume (named pipes)

O conductă cu nume este un fișier temporar ce permite comunicația între procese prin mecanismul `pipe` (conductă). Astfel, procesul emitent scrie date în conducta cu nume, iar procesul receptor citește datele din aceasta. Procesarea datelor din conducta cu nume se face pe baze FIFO.



Existența unui astfel de fișier este limitată la intervalul de timp în care procesele comunică.

Recunoașterea conductelor cu nume se face datorită caracterului `p` cu care începe informația aferentă permisiunilor lor de acces, informație afișată la executarea unei comenzi `ls -l`:

```
prw----- 1 root admin 0 May 12 22:02 nume_conducta
```

Fișiere de dispozitiv

O particularitate care diferențiază sistemul de operare UNIX de alte sisteme, o constituie asocierea dispozitivelor periferice cu fișiere de dispozitiv (fișiere speciale). Fișierele de dispozitiv sunt citite/scrie, din punctul de vedere al utilizatorului, exact ca și cele normale, rezultatul unei astfel de operații fiind activarea driver-ului asociat perifericului respectiv. Un

program de aplicație poate deci utiliza aceeași sintaxă pentru a accesa un fișier normal sau un fișier de dispozitiv. Fie, de exemplu, comanda de copiere a fișierului `test.p`:

```
cp test.p /usr/diploma/copie.p
```

Prin lansarea acestei comenzi se realizează copierea fișierului `test.p` în fișierul destinație `/usr/diploma/copie.p`. Dacă numele fișierului destinație se înlocuiește cu un nume de fișier de dispozitiv, de exemplu imprimanta (`/dev/lp0`), obținem o listare a fișierului `test.p`.

```
cp test.p /dev/lp0
```

Unui dispozitiv periferic îi este asociat cel puțin un fișier de dispozitiv, care va conține întotdeauna informații despre driver-ul (programul de comandă) al aceluși periferic. Toate fișierele de dispozitiv rezidă în directorul `/dev`.

Spre exemplu, fiecărei unități de disc îi corespunde câte un fișier în directorul `/dev`. În Linux, primei unități de dischetă îi corespunde fișierul `/dev/fd0`, celei de-a doua `/dev/fd1`, ș.a.m.d. Primei unități de hard disc cu interfață IDE, din sistem, îi corespunde fișierul special `/dev/hda`, iar primei sale partiții, fișierul `/dev/hda1`. A doua partiție de pe primul disc are ca și corespondent fișierul `/dev/hda2`, a doua unitate de hard disc IDE este referită ca fiind `/dev/hdb`, ș.a.m.d.

În funcție de unitatea de informație pe care o transmit/recepționează la un moment dat, dispozitivele periferice se împart în două tipuri:

- dispozitive orientate pe caractere (terminale, imprimante)
- dispozitive orientate pe blocuri (unități de disc)

Evident, funcție de această caracteristică, dispozitivelor în cauză le corespund fișiere de dispozitiv specifice.

În cadrul unui sistem pot exista mai multe dispozitive periferice de același tip. Pentru a distinge între ele, UNIX-ul asociază fiecărui dispozitiv, două numere:

- un număr major, care indică tipul dispozitivului (de exemplu, imprimanta)
- un număr minor, care indică al câtelea dispozitiv periferic este în clasa de dispozitive de același tip (de exemplu, a doua imprimantă)

Fișierele de dispozitiv caracter se găsesc localizate în directorul `/dev` și furnizează un mecanism pentru comunicarea, un caracter la un moment dat, cu drivere-le dispozitivelor din sistem.

Informația afișată de o comandă `ls -l` în cazul unui astfel de fișier, este similară cu următoarea:

```
crw-rw-rw- 1 root wheel 21, 4 May 12 13:40 ptyp4
```

Se observă că permisiunile de acces în acest caz încep cu caracterul `c`.

Fișierele de dispozitiv bloc se găsesc, de asemenea, localizate în directorul `/dev`, fiind folosite pentru a comunica cu drivere-le de dispozitiv. Ele sunt asociate însă, unor dispozitive periferice ce transferă blocuri mari de date la un moment dat (cum este hard discul).

Aceste fișiere sunt identificate de către caracterul `b` ce apare în fața permisiunilor de acces, atunci când se tastează comanda `ls -l`:

```
brw----- 2 root staff 16, 2 Jul 29 2006 fd0c
```

Crearea fișierelor de dispozitiv, bloc sau caracter, cât și a conductelor cu nume, se poate realiza utilizând comanda `mknod`. Sintaxa acestei comenzi este:

```
mknod fisier_dispozitiv b|c|p|u major minor, unde:
```

`b` semnifică faptul că `fisier_dispozitiv` este un fișier asociat unui dispozitiv bloc

`c` și `u` semnifică faptul că `fisier_dispozitiv` este un fișier asociat unui dispozitiv caracter

`p` semnifică faptul că `fisier_dispozitiv` este un fișier asociat unei conducte cu nume

Unul dintre aceste argumente trebuie să fie prezent în sintaxa comenzii. Este necesară, de asemenea, specificarea numerelor major și minor (cu excepția cazului când se creează o conductă cu nume). Ștergerea fișierelor create cu comanda `mknod` se face cu ajutorul comenzii `rm`.

CAP.4. PROIECTAREA SISTEMELOR INFORMATICE

4.1. SISTEME INFORMATICE

Din punct de vedere funcțional, sistemul informatic are mai multe definiții, în funcție de scop, elementele din structură și relațiile dintre elemente; cea mai generală definiție îl prezintă ca pe *un sistem de colectare, memorare, prelucrare și distribuire a informațiilor care utilizează calculatorul electronic*.

Fiecărui sistem informatic i se asociază un sistem de prelucrare a datelor, în care datele se prezintă pe diferiți suporti de memorare, iar procesele de prelucrare sunt concretizate în proceduri (automate și manuale) executate de diferite echipamente de tehnică de calcul și teletransmisie și de către personal specializat.

Punând în evidență faptul că utilizarea tehnicii de calcul a produs modificări majore în modul de realizare a activităților informaționale, Gh. Sabău și I. Lungu definesc sistemul informatic ca fiind *un ansamblu de elemente interconectate funcțional în scopul automatizării obținerii informației și fundamentării deciziilor*²⁰.

Aceiași autori identifică în structura sistemului informatic următoarele elemente:

1. Baza tehnică (hardware-ul sistemului informatic), cuprinde totalitatea mijloacelor tehnice de culegere, transmitere, prelucrare și stocare a datelor.

2. Sistemul de programe (software-ul), se referă la totalitatea programelor necesare funcționării sistemului informatic în conformitate cu funcțiile și obiectele ce i-au fost stabilite, respectiv programele de bază și programele aplicative.

3. Baza științifico-metodologică este constituită din sistemul indicatorilor economici, procese și fenomene economice, precum și din metodologiile de realizare a sistemelor informatice.

4. Baza informațională cuprinde datele supuse prelucrării fluxurilor informaționale, sistemele și nomenclatoarele de coduri.

5. În resursele umane se include personalul implicat cu funcționarea sistemului informatic, iar **cadru organizatoric** este cel specificat în regulamentul de organizare și funcționare al organismului economic în care funcționează sistemul informatic.

Sistemul informatic al unei întreprinderi este definit de Robert Reix ca fiind *elementul care înglobează toate componentele automatizate ale căror interacțiuni sunt la nivel informațional. Obiectivul său constă în furnizarea la diferite niveluri de organizare a informațiilor ce pot însoți și controla funcționarea întreprinderii*²¹.

Există alte opinii potrivit cărora sistemul informatic poate fi avut în vedere atât ca model extern, cât și sub aspectul unui model intern²².

Modelul extern al sistemului informatic se referă la arhitectura funcțională orientată către utilizator, pe care îl interesează informațiile și procesele lor de prelucrare.

Modelul intern al sistemului informatic se referă la structura fizică orientată către echipamente, unde interesează datele, procedurile de prelucrare, echipamentele, mediile de programare și sistemele de operare.

Tipuri de sisteme informatice. Sistemele informatice pot fi clasificate în funcție de diferite criterii²³.

I. Clasificarea în concordanță cu nivelurile organizatorice. Organizațiile sunt compuse din componente cum ar fi departamente, divizii, echipe etc. Aceste departamente raportează unui nivel organizatoric mai înalt cum ar fi o divizie sau un sediu central într-o

²⁰ Sabău, Gh., Lungu, I. - *Sisteme informatice și baze de date*, ASE, București, 1993, p. 229

²¹ Reix, R. - *Systemes d'information et management des organisation*, Vuibert, Paris, 1995, p.46;

²² Turbuș, Gh. (coordonator) - *Inginerie de sistem, automatizări și informatică în transporturi*, vol. 2, Editura Tehnică, București, 1989

²³ Turban, E., McLean, E., Wetherbe, J., - *Information Tehnology for Management*, John Wiley & Sons, New-York, 1996

manieră ierarhică. Aceasta este o structură ierarhică tradițională cu mai multe niveluri.

O cale pentru organizarea sistemului informatic este construirea lui în conformitate cu structura organizatorică. Aceste sisteme pot fi independente sau pot fi interconectate. De exemplu, un departament de resurse umane poate fi atât la nivel central cât și în fiecare divizie sau sucursală. Sistemele informatice construite conform cu nivelurile organizatorice sunt:

1. Sisteme informatice departamentale. Frecvent, o organizație folosește mai multe programe aplicative într-o singură arie funcțională. De exemplu în gestiunea personalului, este posibil să se folosească un program pentru selectarea solicitanților de locuri de muncă și un program pentru monitorizarea fluctuației personalului. Unele aplicații sunt complet independente în timp ce altele sunt interconectate. Colecția de programe aplicative în domeniul resurselor umane se poate numi *Sistem informatic pentru resurse umane*.

2. Sisteme informatice organizaționale. În timp ce un sistem informatic departamental este în mod obișnuit legat de o arie funcțională, se poate vorbi frecvent de colecții de aplicații în mai multe sau în toate ariile funcționale. O astfel de colecție poate fi descrisă ca un sistem informatic organizațional.

3. Sisteme informatice interorganizaționale. Unele sisteme informatice sunt foarte complexe și implică mai multe organizații. De exemplu sistemul de rezervare a biletelor de avion în lumea întreagă este compus din mai multe sisteme aparținând diferitelor companii aeriene. Deci sistemele informatice interorganizaționale sunt formate din mai multe sisteme informatice organizaționale la care se adaugă sistemele de legătură dintre acestea.

II. Clasificarea în concordanță cu ariile funcționale. După cum s-a observat, sistemele informatice departamentale sunt construite în concordanță cu ariile funcționale tradiționale:

1. Sistemul financiar-contabil,
2. Sistemul pentru producție,
3. Sistemul pentru activitatea comercială,
4. Sistemul pentru gestiunea resurselor umane,
5. Sistemul pentru management.

În fiecare arie funcțională, sunt efectuate operații de bază computerizate care sunt esențiale pentru activitatea organizației. Pregătirea unui ștat de plată sau înregistrarea unui client sunt exemple tipice. Asemenea operații sunt numite *operații scop central* (mission-central tasks). Sistemele informatice care suportă asemenea operații se numesc sisteme tranzacționale (TPS). Sistemele tranzacționale execută operații care se efectuează în toate ariile funcționale dar în special în contabilitate și finanțe.

III. Clasificarea în concordanță cu suportul furnizat de sistem. A treia modalitate de a clasifica sistemele informatice este în concordanță cu tipul suportului pe care îl furnizează, indiferent de aria funcțională implicată. De exemplu, un sistem informatic poate fi un suport pentru funcționari în aproape orice arie funcțională. Similar, managerii, indiferent unde lucrează, pot utiliza un sistem computerizat ca suport pentru luarea deciziilor.

Principalele sisteme după această clasificare sunt:

1. Sisteme tranzacționale (TPS - Transaction Processing System);
2. Sisteme manageriale (MIS - Management Information System);
3. Sisteme automatizate pentru birouri (OAS - Office Automation System);
4. Sisteme suport pentru grup (GSS - Group Support System);
5. Sisteme suport pentru decizii (DSS - Decision Support System);
6. Sisteme executive sau sisteme suport (EIS - Executive Information System or Support System), utilizate în activitatea de execuție;

7. Sisteme suport inteligente (ISS - Intelligent Support System), cuprind sisteme expert (ES - Expert Systems) și rețele neuronale artificiale (ANN - Artificial Neural Networks). Implementarea unei aplicații specifice poate implica două sau mai multe din sistemele prezentate mai sus. De exemplu, un sistem suport pentru decizii poate fi combinat cu un sistem expert pentru a realiza un sistem informatic care să fie utilizat în realizarea unui program promoțional de marketing.

IV. Clasificarea în concordanță cu arhitectura sistemelor informatice - depinde de scopul pentru care a fost construit. Arhitectura și infrastructura sunt două aspecte ale sistemelor informatice ce se interconstrucionează reciproc.

Sistemele informatice pot fi clasificate în concordanță cu arhitectura lor și principalele categorii sunt:

1. Sisteme informatice bazate pe mainframe-uri;
2. Sisteme informatice bazate pe PC-uri;
3. Sisteme informatice distribuite.

V. Clasificarea după natura activităților pentru care sunt construite sistemele informatice. Potrivit acestei clasificări sistemele informatice se împart în:

1. sisteme operaționale;
2. sisteme tactice (manageriale);
3. sisteme strategice.

Sistemele operaționale oferă suportul necesar operațiilor zilnice ale unei organizații cum ar fi înregistrarea unei facturi sau a numărului de ore pe care le lucrează un muncitor. De asemenea, acestea oferă suport și deciziilor operaționale, decizii care se referă la un interval de timp scurt. Sistemele informatice necesare activităților și deciziilor operaționale sunt în general cele tranzacționale, executive precum și sistemele informatice suport pentru decizii. Sistemele operaționale sunt utilizate de către conducătorii de la nivelele joase, operatori și funcționari.

Sisteme tactice (numite și **sistemele manageriale**) sunt legate de activitățile efectuate de către managerii de la nivelele medii ale întreprinderii, activități precum planificarea organizarea și controlul pe termen scurt și mediu. Sistemele tactice față de cele operaționale au o sferă mai largă de cuprindere. Sistemele manageriale se bazează în general pe sistemele informatice manageriale, dar și pe sistemele informatice suport pentru decizii, sisteme informatice executive și chiar sisteme informatice inteligente.

Sistemele strategice sunt legate de situațiile pe termen lung și de deciziile care schimbă semnificativ maniera în care vor fi realizate obiectivele firmei. În mod tradițional sistemele strategice implică doar planificarea de lungă durată. Cele mai utilizate sisteme informatice ca suport pentru sistemele strategice sunt cele suport pentru luarea deciziilor și sistemele expert.

4.2. METODOLOGII DE REALIZARE A SISTEMELOR INFORMATICE

Realizarea sistemelor informatice este o activitate complexă care îmbină un mare **număr de activități eterogene**: analiză, proiectare, programare cu un pronunțat caracter creativ și la care cooperează mai multe unități organizatorice, ocazionând eforturi materiale, financiare și umane pe o perioadă îndelungată.

Folosirea eficientă a resurselor și necesitatea realizării unor sisteme informatice performante au repus ordonarea într-o succesiune bine stabilită pe etape și subetape a acestui proces determinând conturarea unor metodologii de realizare a sistemelor informatice.

Prin metodologia de realizare a sistemelor informatice se stabilesc: **componentele procesului de realizare**:

- etapele, subetapele, activitățile și conținutul lor;
- fluxul executării componentelor;
- metodele, tehnicile și instrumentele de realizare.

4.2.1. Etapele de realizare a sistemelor informatice

Sistemul informatic are un **ciclu propriu de viață**, care începe cu decizia de realizare, cuprinde faza de elaborare, faza de utilizare, faza de perfecționare și se încheie cu decizia de abandonare în forma existentă și înlocuirea cu un nou sistem.

Acestui ciclu de viață îi corespund etape specifice stărilor succesive prin care trece sistemul informatic, etape caracterizate prin activități distincte. **Etapele realizării unui sistem informatic sunt:**

- analiza sistemului informațional existent (analiza de sistem);
- proiectarea sistemului informatic;
- elaborarea și testarea programelor;
- implementarea sistemului informatic;
- exploatarea curentă și menținerea în funcțiune a sistemului informatic.

1) Analiza sistemului informațional existent urmărește delimitarea ariei de cuprindere a sistemului și formularea cerințelor și restricțiilor globale de realizare. Pentru a atinge acest scop, în această etapă se face un studiu amănunțit al sistemului existent, se apreciază măsura în care sistemul existent este capabil să răspundă în continuare exigențelor conducerii științifice a agentului economic, se apreciază oportunitatea realizării unui sistem informatic și se formulează principalele restricții și cerințe pentru viitorul sistem informatic.

2) Proiectarea constă în definirea modelului de ansamblu (conceptual) al sistemului informatic, ținând seama de evaluările făcute în etapa anterioară, dar și în transformarea modelului conceptual stabilit anterior într-un model tehnic, operațional.

În acest scop se proiectează ieșirile noului sistem, se determină entitățile bazei informaționale de intrare, se codifică atributele și se proiectează fluxul general de prelucrare a datelor în noul sistem. De asemenea, se definitivează soluția în organizarea datelor (fișiere sau baze de date), se proiectează fișierele sau bazele de date, se determină și se proiectează procedurile (programele de aplicație) pentru crearea, actualizarea și exploatarea structurilor de date în vederea realizării obiectivelor stabilite sistemului informatic.

3) Elaborarea programelor are la bază soluțiile stabilite în proiectare și urmărește scrierea și testarea individuală a programelor utilizând medii și tehnici de programare adecvate.

4) Implementarea sistemului informatic realizat constă în verificarea modului de comportare practică a modelului proiectat și realizat în vederea trecerii lui în exploatare curentă.

5) Exploatarea curentă și menținerea în funcțiune urmărește atât îndeplinirea obiectivelor inițiale ale sistemului informatic cât și adaptarea acestuia la modificările intervenite în cerințele informaționale ale beneficiarului.

Realizarea unui sistem informatic se concretizează și sub forma unui proiect de sistem informatic prin intermediul căruia se definesc într-o formă standardizată soluțiile adoptate.

4.2.2. Ordinea executării etapelor

Dacă etapizarea procesului de realizare a sistemelor informatice, deși controversată, este în general stabilită, cu privire la ordinea parcurgerii etapelor există diferite orientări generate de diversitatea sistemelor informatice și a realizatorilor lor.

Aceste orientări sunt concretizate în:

- modelul liniar;
- modelul cu prototip;
- modelul cu extensii;
- modelul documentației anticipate;
- modelul ierarhic;
- modele mixte.

Modelul liniar este un model teoretic și presupune parcurgerea secvențială a etapelor cu eventuale reveniri doar la etapa precedentă. Este aplicat doar pentru sisteme informatice de mică anvergură, dar în realitate parcurgerea etapelor este un proces iterativ, desfășurându-se în paralel cu mai multe activități din etape diferite de realizare.

Modelul cu prototip se aplică atunci când se ia decizia elaborării complete, rapide și la costuri scăzute a unei versiuni inițiale, simplificate cu caracter de prototip pe baza căreia se

stabilesc noi specificații de definire a sistemului informatic și se desfășoară activități de realizare a unei noi versiuni a sistemului informatic. Elaborarea noii versiuni se poate face prin parcurgerea integrală sau parțială a etapelor și se modifică numai anumite părți din prototip.

Modelul cu extensii pornește de la premisa că sistemele informatice se pot realiza și da în funcțiune parțial pe aplicații. Realizarea lor se face în manieră extensibilă astfel încât o primă versiune să includă componentele de bază, celelalte urmând să fie realizate și integrate prin extensii ulterioare. Se realizează astfel reducerea eforturilor și fructificarea experienței câștigate în elaborarea componentelor inițiale, deci rezultă că extensiile se ramifică din proiectarea generală.

Modelul documentației anticipate presupune întocmirea în devans a documentației prin parcurgerea etapelor de proiectare, deci el permite analiza critică a documentației urmată de reformularea unor probleme și aspecte înainte de darea în exploatare a sistemului.

Modelul ierarhic se caracterizează prin structurarea ierarhică coerentă a activităților corespunzătoare fiecărei etape, conținutul acestora fiind conform specificului fiecărui nivel.

Modele mixte. În activitatea practică se întâlnesc combinații ale modelelor de parcurgere a etapelor anterior prezentate, adaptate condițiilor concrete de realizare a sistemelor informatice.

O metodologie este o implementare a ciclului de viață al sistemului informatic care face referire la:

- activitățile din fiecare etapă și fază și ordinea lor de desfășurare;
- rolul individual și de grup pe care îl are fiecare activitate;
- condițiile de livrare și standardele de calitate pentru fiecare activitate;
- instrumentele și tehnicile care vor fi utilizate de fiecare activitate.

Există mai multe metodologii de realizare a sistemelor informatice, din care cele mai frecvent utilizate la noi în țară sunt:

- A. SSADM, care se bazează pe analiza structurată și pune accentul pe prelucrări;
- B. MERISE;
- C. OMT, care integrează datele și prelucrările pentru a crea obiecte care pot fi ușor adaptate și reutilizate;
- D. Prototipizarea, care pune accentul pe realizarea și asamblarea de prototipuri.

A. Elemente de bază ale metodologiei SSADM.

SSADM (Structured System Analysis and Design Methodology) este o metodologie structurată de analiză și proiectare a sistemelor informatice. A fost dezvoltată în Marea Britanie, și se bazează pe trei principii metodologice după cum urmează:

- folosirea modelării descendente, de tip top-down, ca mod de abordare; acest mod de abordare permite să se pornească de la cerințele conducerii și să se detalieze cerințele până la nivelul utilizatorilor finali;
- considerarea datelor și prelucrărilor egale ca importanță;
- folosirea unor simboluri grafice unice pe toată durata de realizare a sistemului.

Metodologia SSADM se concentrează cu precădere pe acele etape din ciclul de viață care se referă la elaborarea sistemului: analiza sistemului existent; proiectarea logică; proiectarea fizică.

Tehnicile de analiză și proiectare specifice acestei metodologii sunt: diagrama de flux a datelor; modelul entităților; matricea entități-funcțiuni; normalizarea; structurarea ierarhică a prelucrărilor; documentarea.

Dacă se utilizează această metodologie, echipa de realizare trebuie să acorde atenție deosebită fiecărei faze și să întocmească documentația pentru fiecare etapă, fapt care conduce la micșorarea duratei de realizare și a costului proiectului.

B. Elemente ale metodologiei MERISE.

Această metodologie elaborată în Franța de Ministerul Industriei. Are la bază un model tridimensional care structurează componentele metodologice pe trei cicluri:

- ciclul de abstractizare;
- ciclul de decizie;

- ciclul de viață al sistemului informatic.

Ciclul de abstractizare se dezvoltă, conform modelului ANSI/SPARC de definire a bazelor de date, pe trei niveluri:

- nivelul conceptual,
- nivelul logic,
- nivelul fizic,

cu precizarea că modelul datelor se dezvoltă independent de modelul prelucrărilor.

În cadrul acestui ciclu se realizează succesiv modelul conceptual al datelor și modelul conceptual al prelucrărilor, apoi se elaborează modelul logic al datelor și modelul logic al prelucrărilor (sau modelul organizațional) și apoi se dezvoltă modelul fizic al datelor și modelul fizic al prelucrărilor (sau modelul operațional).

Ciclul de decizie se compune din mulțimea deciziilor ce se iau pe durata ciclului de viață al sistemului informatic, aceste decizii fiind structurate pe tipuri de decizii astfel:

- decizii generale,
- decizii organizatorice,
- decizii tehnice,
- decizii funcționale.

C. Elemente ale metodologiei OMT (Object Modeling Technique)

Creată de către James Rumbaugh, este cea mai cunoscută și utilizată metodologie de proiectare orientată pe obiecte (POO). Dorința de a construi produse software prin asamblarea de componente, adică în aceeași manieră în care se construiește hardware-ul, a stat la baza metodelor POO. Astfel, un sistem informatic este privit ca un ansamblu de obiecte care cooperează între ele, iar obiectele sunt tratate ca instanțe ale unei clase în interiorul unei ierarhii. Orientarea pe obiecte forțează ca toate structurile de date ale unui program să fie încapsulate.

Etaple parcurse în cadrul metodologiei OMT sunt:

1. definirea problemei,
2. elaborarea unei modalități informale de identificare a datelor și funcțiilor relevante din cadrul problemei,
3. formalizarea strategiei prin:
 - a) identificarea obiectelor și a atributelor lor,
 - b) identificarea operațiilor asupra obiectelor,
 - c) stabilirea instanțelor,
 - d) implementarea prelucrărilor.

Metodologia OMT folosește pentru proiectarea sistemelor informatice ca mijloace de reprezentare diagrama de flux de date pentru surprinderea comportamentului dinamic și diagrama modulară pentru surprinderea comportamentului static.

Printre avantajele acestei metodologii se pot enumera:

- produsul informatic este construit din componente care au o calitate probată în implementările anterioare;
- un obiect poate fi modificat fără a afecta celelalte obiecte și deci sistemul informatic astfel construit este modular și poate fi ușor dezvoltat și modificat;
- prin reutilizarea componentelor existente se asigură un timp mai scurt și costuri mai reduse în procesul de realizare a sistemului informatic.

4.2.3. Metode și tehnici de realizare a sistemelor informatice

În activitatea practică au apărut și s-au dezvoltat două tipuri de strategii de abordare și realizare a sistemelor informatice: în funcție de rolul sistemelor informatice în cadrul agentului economic și în funcție de modul de abordare a subsistemelor și aplicațiilor ce compun sistemul informatic.

A. În ceea ce privește strategia de realizare a unui sistem informatic, în funcție de **rolul sistemelor informatice** în cadrul agentului economic rezultă **trei categorii de strategii**: ameliorative; inovatoare; adaptive.

Strategiile **ameliorative** urmăresc automatizarea activităților și operațiilor de rutină sau cu caracter repetitiv. Sistemele informatice ce rezultă nu antrenează mari modificări în sistemul agentului economic. Ele au grad redus de complexitate, nu folosesc personal numeros, timpul de realizare și implementare este scurt, dar sistemul este lipsit de flexibilitate.

În cazul strategiilor **inovatoare**, introducerea sistemului informatic trebuie însoțită de schimbări importante în organizarea și funcționarea agentului economic. Aceste strategii asigură valorificarea superioară a calculatorului electronic, dar presupun un timp mai mare pentru conceperea și realizarea noului sistem, cheltuieli sporite și personal de înaltă calificare.

În cazul strategiilor **adaptive**, sistemul informatic este conceput și realizat astfel încât să fie compatibil cu schimbările care apar în cerințele informaționale și organizatorice și în funcționarea agentului economic. La baza acestor strategii stă principiul invariației sau modificării lente impuse de existența proceselor și structurii de bază ale agentului economic. Elementele informaționale invariante sunt surprinse în baza informațională care ocupă locul central în sistemul informatic. Aceste strategii au avantajul unei flexibilități ridicate datorită conceptului de bază informațională.

Prin comparație, strategiile ameliorative pleacă de la un set bine precizat de ieșiri în funcție de care se determină celelalte elemente ale sistemului informatic: bazele de date, algoritmi, proceduri, iar strategiile adaptive pleacă de la determinarea setului de intrări strict necesare asigurării bazei informaționale astfel încât acestea să modeleze cât mai fidel sistemul agentului economic.

Baza informațională se transpune în colecții de date, proceduri de creare și actualizare care constituie nucleul sistemului informatic. Acesta dă posibilitatea modificării cerințelor informaționale astfel încât să se asigure reglarea fenomenelor și proceselor economice.

B. În funcție de **modul de abordare** a subsistemelor și aplicațiilor ce compun sistemul informatic pot exista **trei tipuri de strategii**: descendentă, ascendentă și mixtă.

Metoda descendentă TOP-DOWN este impusă de faptul că proiectarea sistemelor informatice de mare complexitate face necesară descompunerea ierarhică prin modularizare ca modalitate de control a complexității. Ea constă în descompunerea unui sistem complex pe niveluri ierarhice, succesiv, până la module elementare, simple și relativ independente care sunt controlate de module coordonatoare.

Metoda are ca cerință de aplicare modularizarea sistemului, deci obiectivul principal este realizarea modularizării de sus în jos, iar din obiectivul principal rezultă următoarele obiective specifice: crearea posibilității de realizare în paralel a componentelor sistemului informatic și eliminarea din sistem a redundanțelor.

Modul de lucru în cadrul acestei metode este următorul :

- se realizează mai întâi o descompunere a sistemului în funcții principale și se stabilesc relațiile dintre acestea, aceste funcții stând la baza unei prime modularizări;
- se face apoi analiza modulelor obținute și dacă se identifică noi funcții se trece la descompunerea pe următorul nivel;
- procesul descompunerii continuă până toate modulele sunt terminale (un modul este terminal când nu se mai poate descompune în alte module).

Descompunerea are la bază următoarele reguli:

- nivelul 0 sau punctul inițial de pornire îl constituie un modul neterminal sau coordonator;
- pentru toate modulele neterminale ale sistemului se aplică descompuneri succesive, în pași, de sus în jos;
- descompunerea este terminată când modulele ultimului nivel sunt terminale.

Aplicarea în practică prezintă dezavantaje generate de defnirea modelului de ansamblu a sistemului informatic pe baza unei analize complexe cu personal numeros, ceea ce determină

prelungirea termenului de dare în exploatare a sistemului, iar erorile în definirea structurii și a relațiilor dintre module pot afecta toată activitatea ulterioară.

Metoda ascendentă BOTTOM-UP constă în agregarea modulelor de jos în sus punând în evidență legăturile dintre ele până se ajunge la un singur modul.

Conceptele care stau la baza acestei metode sunt ca și la metoda descendentă: modularizarea și abordarea sistemică.

Regulile care stau la baza metodei sunt:

- nivelul de agregare inițial este nivelul la care se află modulele terminale;
- agregarea se face succesiv de jos în sus;
- când se obține un nivel de agregare se realizează integrarea modulelor de nivel inferior în module de nivel superior;
- agregarea este terminată când la un nivel de agregare se obține un singur modul.

Aplicarea acestei metode prezintă avantajul că sistemul se dezvoltă treptat, în concordanță cu cerințele reale ale utilizatorilor. Agentul economic poate beneficia mai repede de rezultatele prelucrării automate a datelor, se familiarizează cu sistemul în mod gradat, se reduc riscurile realizării unor sisteme de mare anvergură, neoperaționale.

Dezavantajele acestei metode rezultă din gradul de integrare redus a modulelor datorită lipsei unei concepții inițiale de ansamblu, ceea ce face necesară reproiectarea unor componente.

4.3. ANALIZA SISTEMULUI INFORMAȚIONAL EXISTENT

4.3.1. Conceptul de analiză a sistemelor informaționale

În general, analiza se definește ca fiind descompunerea reală sau mintală a unui obiect, fenomen sau a relațiilor dintre obiecte, fenomene etc, în părți componente în scopul cunoașterii. Ea presupune deci examinarea amănunțită, parte cu parte, a unei probleme, a unui obiect de studiu.

Analiza sistemelor informaționale economice este activitatea prin care se realizează cunoașterea sistemului obiect și a cerințelor de informații din sistemul de decizie. Vom numi sistem obiect orice parte a realității (a sistemului informațional economic, în cazul nostru) care generează date și care posedă entități, ce poate prelucra și atribui semnificație datelor, parte delimitată în vederea realizării unui nou sistem informațional.

Scopul analizei sistemului informațional ar putea fi unul din următoarele:

- 1) proiectarea unui model al sistemului existent, adică a unui sistem sinonim cu ajutorul căruia să se simuleze funcționarea sistemului real;
- 2) proiectarea unui sistem care să înlocuiască sistemul existent, având însă performanțe superioare;
- 3) proiectarea unui sistem raționalizat:
 - a) care pe baza aceluiași intrări elaborează aceleași ieșiri, dar cu o structură mai simplă; de exemplu se raționalizează structura organizatorică în paralel cu raționalizarea sistemului de evidență;
 - b) care simplifică intrările, structura și în cazul în care se dovedește necesar, ieșirile sistemului.

Există două concepte care stau la baza stabilirii elementelor sistemului obiect, care aplicate la activitatea de analiză identifică două tipuri de structuri:

- conceptul orientat pe organism, care determină structura organizatorică; în acest caz analiza sistemului informațional se va face pe compartimente funcționale;
- conceptul orientat pe activitate, care determină structura funcțională; caz în care analiza sistemului informațional se va face pe funcții și subfuncții ale agentului economic analizat.

În general sunt studiate ambele tipuri de structuri, accentul punându-se însă pe activități, în timp ce elementele organizatorice interesează doar prin prisma aportului lor la realizarea activităților.

Activitatea de analiza a unui sistem informațional economic se poate descompune în următoarele operații:

- culegerea datelor
- sistematizarea datelor culese;
- evaluarea sistemului informațional existent.

În vederea delimitării corecte a cerințelor informaționale se pot distinge două variante de abordare a sistemului obiect, și anume de la analiza deciziilor de la analiza datelor:

Abordarea pornind de la analiza deciziilor caută să determine cerințele de informații pornind de la analiza obiectivelor și a deciziilor ce trebuie luate. Sunt culese și investigate doar acele informații care sunt necesare modelului de decizie, și anume:

- identificarea obiectivelor și a deciziilor curente și potențiale, precum și a atributelor conducerii, corespunzătoare obiectivelor și deciziilor;
- identificarea sau construirea unui model, a unei proceduri de elaborare a fiecărei decizii sau a unui model al procesului de luare a deciziilor;
- testarea sensibilității modelului la acuratețea și disponibilitatea datelor de intrare, specificarea limitelor și disponibilităților datelor cerute de model.

Avantajul acestui mod de abordare este acela că rezulta un volum mic de date de prelucrat.

În abordarea pornind de la analiza datelor se caută să se obțină cerințele informaționale prin analiza datelor folosite curent în sistemul existent, sau a datelor potențial folosibile.

Sunt identificate datele colectate din sistemul existent, pentru care s-a perceput o necesitate, cât și datele necolectate în sistemul existent, dar semnalate ca fiind utile, și anume:

- cercetarea tuturor documentelor, fișierelor folosite în mod curent și identificarea datelor care sunt culese și prelucrate în mod curent;
- identificarea datelor suplimentare, care nu sunt culese și prelucrate în mod curent, dar care se dovedesc a fi utile;
- stabilirea datelor inutile, care se culeg în mod curent și nu sunt utilizate.

Acest mod de abordare pleacă de la premisa că toate datele utile trebuie să facă parte din sistem, pentru a preveni schimbările ce pot surveni în mediul agentului economic studiat, în stilul de decizie sau în cerințele informaționale.

Abordarea pornind de la analiza datelor conduce la un volum mai mare de date, dar are avantajul că deciziile ce se vor lua pe baza acestor date vor fi mai puțin afectate de modificările ce pot surveni pe parcurs.

Fiecare dintre cele două abordări este mai potrivită pentru un anumit tip de probleme, aplicații sau sisteme, analiza deciziilor se recomandă în special la activitățile de conducere, în cazul sistemelor informaționale de management și în cazul sistemelor suport de decizie, iar analiza datelor se recomandă în special în cazul activităților de rutină, în cazul sistemelor de gestiune.

4.3.2. Obiectivele și fazele analizei sistemului informațional existent (ASIE)

Această etapă de realizare a unui sistem informatic mai este cunoscută și sub denumirea de *elaborarea temei de realizare, analiza diagnostic sau studiul de oportunitate*.

Această etapă are următoarele obiective :

1) Delimitarea ariei de cuprindere a sistemului informațional existent care va deveni sistem obiect pentru conceperea și realizarea unui sistem informatic. (Vom numi sistem obiect acea parte a realității economice care generează date și care posedă entități ce pot prelucra și atribui semnificație datelor, parte delimitată în vederea realizării unui sistem informatic).

2) Reflectarea activităților și operațiilor de culegere, transmitere și prelucrare a datelor specifice sistemului informațional existent.

3) Surprinderea modificărilor ce se impun în organizarea și funcționarea sistemului informațional existent în viziunea conceperii unui sistem informatic.

4) Evidențierea dotării cu tehnică de calcul, prin prisma concordanței dintre sistemele electronice de calcul și viitorul sistem informatic, inclusiv identificarea problemelor ce urmează a fi rezolvate de noul sistem cu ajutorul calculatorului electronic.

5) Fundamentarea unei soluții de principiu care să precizeze activitățile și operațiile ce vor fi informatizate, costul antecalculat al sistemului, tipul calculatorului electronic, sistemul de operare aferent, inclusiv soluția de gestiune a datelor (sistem de gestiune a fișierelor sau sistem de gestiune a bazelor de date) pe care se va baza conceperea și realizarea noului sistem, implicațiile în organizarea internă a unității, precum și planificarea globală a realizării sistemului informatic.

ASIE este necesară pentru a fundamenta direcțiile de perfecționare a sistemului informațional existent și înlocuirea acestuia cu un sistem informatic care să satisfacă toate cerințele informaționale ale agentului economic. De asemenea analiza de sistem este oportună și în cazul agenților economici nou înființați pentru fundamentarea proiectului de sistem informațional, ca parte a proiectului de înființare a unității.

ASIE trebuie să ofere conducerii agentului economic informațiile necesare fundamentării deciziei de perfecționare a sistemului informațional și de proiectare a unui sistem informatic.

ASIE se desfășoară în următoarele faze :

- 1) **Organizarea și conducerea analizei sistemului informațional prin:** pregătirea condițiilor necesare analizei; constituire colectivului pentru ASIE; elaborarea programului de realizare;
- 2) **Realizarea ASIE prin:** documentarea pentru analiza de sistem; alegerea tehnicilor de analiză a sistemului informațional existent; studiul componentelor sistemului informațional existent; evaluarea critică a sistemului informațional existent; elaborarea variantelor de realizare a sistemului informatic.
- 3) **Finalizarea ASIE prin:** definitivarea documentației ASIE; avizarea ASIE de către beneficiar.

4.3.3. Organizarea și conducerea ASIE

Pregătirea condițiilor necesare ASIE. Inițierea și declanșarea analizei de sistem este atributul conducerii unității beneficiare care precizează tema și obiectivele analizei printr-o Notă de comandă. Pe baza Notei de comandă se încheie Contractul de execuție pentru ASIE între unitatea beneficiară și unitatea de analiză proiectare. Valoarea contractului se determină pe baza unui deviz în care se stipulează: numărul de ore/om preconizat pentru executarea analizei, personalul implicat, tarifele de realizare, termenele de execuție, inclusiv obligațiile părților și modul de decontare a lucrărilor.

Constituirea colectivului pentru ASIE. Colectivul poate fi alcătuit din :

- personalul propriu al unității beneficiare;
- personalul unei unități specializate în analiza și proiectarea de sisteme informatice;
- din personalul unității beneficiare și al unei unități specializate.

Prima variantă prezintă avantajul cunoașterii sistemului informațional de către colectivul de analiză, ceea ce asigură efectuarea unei analize de calitate într-un termen relativ scurt, dar prezintă dezavantajul că membrii colectivului nu pot sesiza toate deficiențele și limitele reale ale sistemului informațional, deoarece sunt familiarizați cu structura și funcționarea acestuia.

Varianta a doua prezintă avantajul că membrii colectivului de analiză sunt specializați în rezolvarea unor astfel de probleme, cunosc deficiențele și limitele unor sisteme similare cu cel analizat, dar prezintă dezavantajul realizării analizei într-un termen relativ îndelungat, deoarece se impune o cunoaștere detaliată a unor anumite părți cu caracter specific, de multe ori unicat, ale sistemului informațional.

Varianta a treia este cea mai indicată datorită maximizării avantajelor și minimizării dezavantajelor primelor două variante.

În orice variantă, pe lângă analiștii de sistem, din componența colectivului trebuie să facă parte conducători și membri ai compartimentelor funcționale supuse analizei. Aceasta deoarece conducătorii compartimentelor funcționale pot identifica informațiile utile, pe cele de prisos și pot formula cerințe informaționale pentru noul sistem, iar personalul de specialitate din compartimentele funcționale, cunoscând detaliat toate activitățile și operațiile desfășurate, poate sesiza neajunsurile segmentului de sistemului informațional pe care lucrează.

Numărul membrilor colectivului depinde de obiectivele analizei, de sfera de cuprindere, de complexitatea și particularitățile sistemului informațional și de termenele de realizare.

Elaborarea programului de realizare. Desfășurarea activității de analiză a sistemului se efectuează pe baza unui program prin care se urmărește folosirea rațională a personalului, termenele intermediare și finale, precum și procedurile de control și avizare a realizării ASIE. Programarea realizării ASIE se poate face prin folosirea grafului de tip PERT iar urmărirea operativă a execuției se poate efectua prin intermediul Graficelor de tip GANTT.

4.3.4. Realizarea analizei sistemului informațional existent

Documentarea pentru ASIE. Activitatea de documentare necesară analizei de sistem presupune studiul unor probleme de organizare ale agentului economi, a particularităților procesului tehnologic, a fluxurilor informaționale și a activităților desfășurate, precum și a cadrului legislativ impus funcționării sistemului informațional.

În această viziune se studiază modul de organizare și conducere a agentului economic; obiectul activității de bază; particularitățile procesului tehnologic; strategia și tactica conducerii în realizarea activităților de bază; principalele acțiuni pentru dezvoltarea unității; etc.

În continuare se studiază actele normative care reglementează activitatea agentului economic în ansamblu și activitatea fiecărui compartiment funcțional precum și modul în care se respectă cadrul legal și deciziile de funcționare efectivă a sistemului informațional al unității analizate.

Documentarea este bine să fie completată cu studiul literaturii de specialitate și a celor mai noi realizări în organizarea și funcționarea sistemelor informaționale, precum și cu experiența pozitivă din unități care fac parte din aceeași ramură de activitate cu unitatea studiată.

Alegerea tehnicilor de analiză. Analiza sistemelor informaționale se poate realiza utilizând mai multe tehnici. Astfel distingem două categorii de tehnici de analiză a sistemelor informaționale economice :

- tehnici elementare;
- tehnici complexe.

Tehnicile elementare realizează doar operația de culegere a datelor și asigură o sistematizare redusă a acestora. În grupa tehnicilor elementare de analiza se încadrează:

- observarea directă;
- studierea documentației existente;
- participarea analistului la efectuarea lucrărilor din sistemul informațional;
- interviul;
- chestionarul.

Tehnicile complexe realizează toate activitățile de analiza, respectiv culegerea și sistematizarea datelor, precum și evaluarea sistemului informațional. Culegerea datelor, în cadrul tehnicilor complexe de analiză, se realizează utilizând, de regulă, mai multe tehnici elementare.

Sistematizarea datelor, constând nu doar într-o simplă ordonare a lor, ci și într-o sintetizare, se realizează folosind una sau mai multe tehnici de reprezentare. Dintre tehnicile de reprezentare amintim:

- scrierea tehnica;

- reprezentarea grafică a fluxurilor informaționale prin: organigrame (scheme logice), schema bloc a fluxurilor informaționale (diagrama de rutina sau diagrama de relații), schema orizontală a fluxurilor informaționale, schema verticală a fluxurilor informaționale;
- grilele;
- listele de acțiuni condiționate;
- tabelele de decizii;
- limbajul pseudocod etc.

Dintre tehnicile complexe de analiza a sistemelor informaționale fac parte:

- analiza diagnostic;
- analiza celulară;
- analiza S.O.P. (Study Organization Plan).

În general, alegerea uneia sau altele din tehnicile elementare de investigare a sistemului informațional existent și din tehnicile de reprezentare se face în funcție de: complexitatea și particularitățile sistemului informațional existent; aria de cuprindere a acestuia; condițiile concrete de lucru; termenele de realizare; experiența analiștilor.

Practica a cristalizat câteva recomandări privind alegerea și utilizarea tehnicilor de analiză:

1) Nici o tehnică nu poate, ea singură, să asigure culegerea tuturor datelor pentru caracterizarea complexă a sistemului informațional existent, de aceea se impune utilizarea mai multor tehnici. De regulă se utilizează ca tehnică principală interviul completat cu chestionarul sau cu inventarul documentelor.

2) În vederea alegerii corecte a tehnicii de culegere a datelor și stabilirii modului de realizare, se impune o bună documentare prealabilă privind unitatea ce va fi supusă analizei, în special personalul său.

3) În culegerea datelor analistul nu va avea idei preconcepute despre problemele unității și despre modul de rezolvare a lor, deoarece, în loc să descopere adevăratul mecanism al activității investigate, va primi doar confirmarea propriilor păreri și construirea proiectului pe această bază va conduce sigur la eșec.

4) În timpul culegerii datelor nu se elaborează soluții, eventualele idei de soluții se notează fără a insista asupra lor (din același motiv ca în cazul anterior).

Studiul componentelor sistemului informațional existent. Analiza elementelor componente ale sistemului informațional următoarele obiective:

- a) identificarea caracteristicilor generale ale agentului economic;
- b) studiul activităților desfășurate de agentul economic;
- c) studiul deciziilor din sistemul agentului economic;
- d) studiul dotării cu tehnică de calcul;
- e) studiul fluxurilor informaționale;
- f) studiul volumului datelor;
- g) studiul costului de funcționare a sistemului informațional.

a) În cadrul primului obiectiv se studiază: profilul și obiectivele agentului economic, cadrul juridic de funcționare, dinamica unor indicatori tehnico-economici privind capacitatea de producție, gradul de folosire a utilajelor, cifra de afaceri, numărul de salariați, metodele de previziune, forma de contabilitate, metoda de evidență a valorilor materiale, metoda de calculație a costurilor, relațiile informaționale ale unității analizate cu alți agenți economici și cu organe ierarhice superioare și inferioare, determinarea modului de ierarhizare a secțiilor de producție și a compartimentelor funcționale prin organigrama structurii organizatorice.

b) În cadrul celui de-al doilea obiectiv se studiază: funcțiile unității pe baza statutului de funcționare: activitățile și sarcinile din cadrul funcțiilor, atribuțiile fiecărui post de lucru, dar și funcția de bază a agentului economic.

c) Studiul deciziilor din sistemul agentului economic urmărește modul cum sunt fundamentate aceste decizii documentele existente și informațiile de bază.

d) Studiul dotării cu tehnică de calcul permite analiza dotării cu tehnică de calcul, gradul ei de utilizare și măsura în care aceasta contribuie la funcționarea sistemului informațional existent.

Se analizează modul în care configurația existentă răspunde cerințelor actuale din punct de vedere al capacității de stocare și prelucrare, precum și posibilitățile de integrare în noul sistem.

e) Studiul fluxurilor informaționale verifică circuitul documentelor având în vedere:

- descrierea fluxurilor informaționale printr-o organigramă unde se precizează: activitățile, compartimentele sau posturile de lucru, procedurile efectuate asupra documentelor și circuitul documentelor între posturile de lucru;
- evidențierea fluxurilor paralele de date și a circuitelor neraționale care trebuie eliminate din noul sistem;
- identificarea documentelor vehiculate în mod inutil sau care circulă într-un număr de exemplare necorespunzător cerințelor reale ale sistemului informațional;
- determinarea gradului de utilizare a documentelor tipizate pentru eliminarea din sistem a documentelor netipizate;
- determinarea gradului de încărcare și solicitare a fiecărui compartiment și post de lucru implicat în sistemul informațional;
- integrarea sistemului informațional analizat cu alte sisteme informaționale.

f) Studiul volumului datelor se referă la evidențierea și cuantificarea următoarelor elemente: simbolul și denumirea documentelor, frecvența de întocmire a documentelor, numărul maxim și mediu de documente corespunzător frecvenței de întocmire a documentelor, numărul maxim și mediu de rânduri pe document, numărul maxim și mediu de caractere pe rând, numărul de exemplare pe document, estimarea numărului mediu de documente pentru perioada viitoare.

Concomitent cu analiza documentelor se urmărește sistemul de coduri pentru a fi preluat în noul sistem informatic deoarece activitatea de codificare este complexă și personalul din unitate este familiarizat cu codurile existente.

Pe baza evaluărilor făcute, echipa de analiză poate formula recomandări privind tipul și capacitatea perifericelor, suporturile tehnice de date necesare în sistem și poate face o evaluare critică a sistemului informațional existent.

g) Studiul costului de funcționare al sistemului informațional se referă la analiza cheltuielilor cu salariile personalului implicat în funcționarea sistemului, a cheltuielilor cu amortizarea și întreținerea tehnicii de calcul, a cheltuielilor cu materiale consumabile. Identificarea costului de funcționare a sistemului informațional existent servește în faza de analiză de sistem pentru a aprecia performanțele acestuia și după implementarea sistemului pentru calculul eficienței economice a noului sistem când se compară cheltuielile cu efectele noului și vechiului sistem.

Evaluarea sistemului informațional. Prin aceasta se urmărește evaluarea performanțelor și limitelor sistemului informațional în raport cu cerințele sistemului de conducere și evaluarea gradului de pregătire a unității analizate pentru realizarea unui sistem informatic.

Evaluarea performanțelor unui sistem informațional se face pe baza următoarelor criterii:

- măsura în care sistemul informațional asigură realizarea obiectivelor și îndeplinirea funcțiilor de bază ale unității;
- gradul de asigurare cu informațiile necesare la diferite niveluri de conducere;
- operativitatea în culegerea, prelucrarea datelor, transmiterea informațiilor și adoptarea deciziilor prin prisma timpului de răspuns;
- calitatea informațiilor rezultate din prelucrare, privită sub aspectul preciziei de exprimare, al flexibilității și frecvenței de difuzare;
- calitatea și siguranța legăturilor între posturile de lucru în cadrul fluxurilor informaționale;
- posibilitățile sistemului informațional de a sesiza tendințele în evoluția unității;
- posibilitățile de control și efectuare de corecții, reacția la apariția unor evenimente externe, posibilitățile de corectare în timp util a abaterilor;

- gradul de integrare a sistemului informațional sub aspectul reducerii informațiilor redundante, a compatibilității informațiilor cu datele de intrare, a organizării datelor în baze de date unitare și posibilitatea de asigurare a unui cadru organizatoric adecvat;
- gradul de automatizare a lucrărilor în totalul lucrărilor din sistemul informațional.

Evaluarea gradului de pregătire a unității analizate pentru proiectarea unui sistem informatic are în vedere: pregătirea personalului, disciplina tehnologică în sistemul informațional, existența cadrului organizatoric, existența bazei științifico-metodologice și informaționale.

Concret, evaluarea sistemului informațional trebuie să facă referiri la:

- obiectivele sistemului;
- mijloacele și metodele de prelucrare;
- personalul implicat;
- costurile de funcționare ale sistemului.

Evaluarea critică a obiectivelor urmărește să determine modul în care sunt rezolvate cerințele informaționale din sistem, măsura în care informațiile solicitate lipsesc sau sunt necorespunzătoare. Trebuie pusă în evidență concordanța dintre structura organizatorică și categoriile de informații necesare fiecărui post. Existența în sistemul informațional a unor informații de prisos este la fel de dăunătoare ca și lipsa de informații, deci, trebuie făcută o inventariere exactă a informațiilor din sistem, evidențiate procedurile asemănătoare de prelucrare a datelor și fluxurile paralele de date.

Evaluarea critică a mijloacelor tehnice utilizate trebuie să pună în evidență gradul de dotare cu tehnică de calcul, ponderea acesteia în sistemul informațional, gradul ei de utilizare și oportunitatea folosirii tehnicii existente în noul sistem. Se poate analiza gradul de flexibilitate al tehnicii de calcul și măsura în care aceasta este compatibilă cu exigențele actuale și previzibile ale sistemului. Se pot face evaluări privind gradul de uzură fizică sau morală a sistemului de calcul.

Evaluarea critică a personalului are rolul de a determina gradul de utilizare al timpului de lucru și gradul de încărcare pe fiecare compartiment și post de lucru. Se poate analiza dacă structura personalului pe profesii și niveluri de pregătire este în concordanță cu exigențele impuse de specificul sistemului informațional. Se pot face aprecieri asupra gradului de perfecționare profesională a personalului.

Concret, se poate determina numărul total de personal implicat în funcționarea sistemului informațional și fondul de timp maxim disponibil alocat, exprimat în ore-muncă, numărul total de documente sau informații vehiculate în sistem raportate la numărul total de personal, ponderea fondului de timp maxim disponibil alocat personalului ce lucrează în sistemul informațional în totalul fondului de timp de la nivelul agentului economic, productivitatea muncii pe persoană, timpul mediu pentru completarea unui document, ponderea salariaților care lucrează în sistemul informațional în totalul salariaților și dinamica acestor indicatori.

Evaluarea costului de funcționare a sistemului informațional presupune punerea în relief a măsurii pentru reducerea cheltuielilor cu funcționarea sistemului informațional. Este indicată precizarea efectelor negative care apar datorită lipsei din sistem a unor informații și estimarea lor valorică. Se pot calcula următorii indicatori cu privire la costul sistemului informațional: cheltuieli cu informațiile strict necesare; cheltuieli cu informațiile suplimentare; cheltuieli necesare pentru utilizarea a "n" informații; cheltuieli cu informațiile la 1000 de lei volum de activitate; costul mediu de utilizare pe un document .

Elaborarea variantelor de realizare a noului sistem. Presupune parcurgerea următoarelor activități:

- I) stabilirea funcțiilor, cerințelor și restricțiilor noului sistem;
- II) definirea soluției globale de principiu pentru prelucrarea automată a datelor;
- II) evaluarea fiecărei variante de realizare.

D) Stabilirea funcțiilor, cerințelor și restricțiilor noului sistem se bazează pe evaluarea critică a sistemului existent și pe cerințele beneficiarului.

Stabilirea funcțiilor noului sistem presupune delimitarea ariei de cuprindere a principalelor activități pe care le va acoperi noul sistem, legătura cu aplicațiile informatice în exploatare sau prevăzute a se realiza în viitor.

Subsistemele, aplicațiile informatice care se pot fi proiectate și realizate într-un sistem economic sunt:

1. Subsistemul pentru activitatea de planificare tehnico-economică:

- elaborarea planului anual: stabilirea variantelor de plan, calculul indicatorilor tehnico-economici, alegerea variantei optime. Se elaborează studii de prognoză privind cerințele pieței, se întocmesc mai multe variante urmărind valorificarea potențialului productiv existent și crearea resurselor necesare pentru realizarea lor.
- defalcarea planului pe trimestre, luni, subunități care se face în funcție de nivelul resurselor, termenele contractuale și prioritățile beneficiarului. Se recalculează valorile indicatorilor economico-financiari.
- urmărirea și analiza realizării planului. Se centralizează pe baza rapoartelor de producție realizările și se semnalează abaterile, se calculează indicatorii economici și se semnalează abaterile.

2. Subsistemul pentru pregătirea tehnică a producției:

- elaborarea și actualizarea nomenclatoarelor de produse pe baza informațiilor din documentația constructivă a produselor;
- elaborarea și actualizarea fișierelor tehnologice pe baza precizărilor din documentația tehnologică pe fiecare produs;
- calculul consumului specific de manoperă pe baza documentației specifice, pe meserii, utilaje, diferite niveluri de agregare a produselor;
- calculul loturilor optime și ciclurilor de fabricație pe baza documentației tehnice;
- calculul consumurilor specifice de materii prime, materiale, SDV;
- urmărirea și analiza încadrării în normele de consum;
- urmărirea și analiza planului privind realizarea produselor noi.

3. Subsistemul de programare, lansare și urmărire a producției de bază:

- calculul necesarului de fabricat pe o perioadă pe baza planului de producție și a nomenclatoarelor de produse;
- calculul fondului de timp al utilajelor pe baza indicatorilor privind starea utilajelor și planul de reparații;
- elaborarea programelor de fabricație trimestriale și lunare în funcție de termenele de livrare, disponibilul de capacități, forța de muncă și resursele materiale;
- programarea operativă a producției, detalierea sarcinilor din programul de fabricație la nivel de decadă, zi, schimb, pe ateliere, grupuri de mașini pentru stabilirea unor loturi optime de fabricație;
- lansarea manoperei care presupune calculul și centralizarea consumului de timp necesar executării operațiilor pentru producție și emiterea bonurilor de manoperă;
- lansarea materialelor, calculul necesarului de materiale, emiterea bonurilor de consum sau a fișelor limită de consum;
- controlul calității producției, urmărirea și analiza realizării producției.

4. Subsistemul pentru aprovizionarea tehnico-materială:

- calculul normativelor de materii prime și materiale;
- planul de aprovizionare pe baza planului de producție și a normelor de consum;
- calculul necesarului pentru aprovizionat cu defalcarea pe trimestre și luni pe baza planului de aprovizionare și a stocului existent la începutul perioadei;
- lansarea și urmărirea comenzilor de aprovizionare;
- urmărirea gradului de acoperire a necesarului de aprovizionat;
- analiza și urmărirea realizării activității de aprovizionare.

5. Subsistemul pentru activitatea de desfacere:

- elaborarea de studii de previziune privind cerințele pieței;
- evidența și analiza comenzilor primite de la beneficiari;
- urmărirea gradului de acoperire a planului de producție cu comenzi și contracte;
- întocmirea planului anual de livrare;
- urmărirea derulării contractelor cu beneficiarii;
- urmărirea și analiza privind activitatea de desfacere.

6. Subsistemul pentru gestiunea valorilor materiale:

- gestiunea stocului de materii prime și materiale;
- gestiunea stocului de produse finite;
- gestiunea stocului de semifabricate proprii;
- urmărirea și analiza stocurilor supranormative și a celor cu mișcare lentă.

7. Subsistemul pentru activitatea de personal:

- elaborarea planului forței de muncă, calculul fondului de salarii pe baza planului de producție, a normelor tehnologice;
- evidența și structura personalului: elaborarea situației privind existentul și prezența la lucru a personalului, fluctuația personalului;
- calculul salariilor;
- analiza și raportarea forței de muncă.

8. Subsistemul pentru activitatea financiar-contabilă:

- elaborarea bugetului de venituri și cheltuieli;
- evidența mijloacelor fixe: urmărirea pe locuri de folosință, calculul și includerea pe costuri a amortizării, urmărirea uzurii fizice și morale;
- contabilitatea valorilor materiale;
- contabilitatea salariilor: întocmirea formelor de plată: lista de avans chenzinal, ștutul de salarii, indemnizații pentru concediul de odihnă, plățile în contul asigurărilor sociale;
- calculația costurilor: înregistrarea corectă și la timp pe locuri de folosință și obiecte de calculație a cheltuielilor de producție, calculul costurilor unitare pe produs, analiza costurilor de producție.
- contabilitatea generală cu: evidența mijloacelor bănești și a împrumuturilor, debitori, creditori, decontările cu furnizorii, cu beneficiarii, cu bugetul de stat, cu asigurările sociale, veniturile și rezultatele financiare; întocmirea bilanței de verificare și a situațiilor de raportare; întocmirea bilanțurilor și anexelor sale.

9. Subsistemul pentru activități auxiliare sau specifice:

- activitatea de întreținere și reparații utilaje;
- activitatea de realizare a pieselor de schimb;
- activitatea de transport.

Restricțiile viitorului sistem reprezintă condiții impuse proiectantului de către beneficiar sau cadru legal. Ele pot fi:

- de natură legislativă: proiectantul trebuie să cunoască actele normative ce reglementează activitatea în sistemul obiect și să țină seama de prevederile acestora în realizarea noului sistem;
- de natură organizatorică: nu există specialiști în informatică și până la formarea lor se apelează la o unitate specializată;
- de natură tehnică: viitorul sistem nu se poate realiza decât cu o configurație de calcul și cu un soft de bază;
- cu caracter general: proiectul trebuie terminat la o anumită dată și cu un anumit cost.

II) Definirea unor soluții globale de principiu pentru prelucrarea automată a datelor are în vedere obiectivele stabilite anterior prin definirea mai multor soluții de prelucrare automată în noul sistem. Pentru fiecare variantă se stabilesc numărul activităților și gradul de

informatizare a acestora, delimitându-se operațiile ce se efectuează automat de cele care se vor efectua manual. Se vor preciza atribuțiile preluate de sistemul informatic în raport cu cele ce vor rămâne pe seama compartimentelor funcționale. Avându-se în vedere aceste opțiuni cu caracter funcțional se studiază soluțiile tehnice de principiu pentru:

a) Posibilitățile financiare ale agentului economic de a dispune de configurația de calcul necesară. Echipamentul poate fi propriu sau închiriat;

b) Felul calculatoarelor: mari, medii, mini, micro și modul lor de folosire. Interconectarea într-o rețea, chiar eterogenă, cu regim de exploatare distribuit local sau la distanță, cu stații de lucru bazate pe calculatoare personale are avantajul creșterii numărului de utilizatori, exploatării intensive a capacității de prelucrare și satisfacerii în timp real a cerințelor utilizatorilor, dar este mai costisitoare;

c) Tipul sistemului de operare și softului de bază minim pentru funcționarea sistemului;

d) Felul produselor-program ce vor fi folosite: pot fi produse-program specifice sau generalizate. Prima soluție are avantajul posibilității de informatizare a problemelor complexe specifice fiecărui beneficiar, dar are dezavantajul că necesită un personal strict specializat. A doua variantă scurtează durata de realizare și costul este relativ scăzut, dar are dezavantajul limitării informatizării la facilitățile oferite de produsul-program și este puțin flexibilă;

e) Tipul suporturilor tehnice de date folosite în noul sistem și modalitatea concretă de introducere a datelor în sistemul de calcul și obținerea rezultatelor. Soluția cea mai eficientă de introducere a datelor este preluarea directă de la terminal. Afișarea rezultatelor se poate face la imprimantă pentru situațiile de ieșire ce vor servi ca documente justificative, dar cu costuri mari și un timp mare de obținere a rezultatelor, sau se poate face la videoterminal;

f) Modul de organizare a datelor:

- în fișiere independente: se realizează ușor, redundanță mare a datelor, limitarea ieșirilor la datele stocate în fișiere, dependența fizică și logică a programelor de aplicație față de modul de descriere a datelor;
- în baze de date gestionate de un sistem de gestiune a bazelor de date;
- în baze de date distribuite local sau la distanță care oferă facilități superioare în actualizarea și exploatarea datelor, dar cu eforturi mai mari;
- în baze de cunoștințe care prezintă avantaj din punct de vedere a posibilităților de prelucrare și dezavantaj din punct de vedere al modului de realizare și al costurilor mari.

g) Tipul de prelucrare a datelor: pe loturi, interactivă, centralizată fără teleprelucrare și centralizată cu teleprelucrare distribuită. Prelucrarea pe loturi este avantajoasă pentru prelucrări multiple efectuate o singură dată la sfârșitul unei perioade de gestiune, dar prezintă dezavantajul lipsei de operativitate a informațiilor. Prelucrarea interactivă are avantajul realizării unor sisteme informatice dinamice, asigură obținerea de informații în timp real, dar în anumite sisteme de calcul nu folosesc intensiv capacitatea de prelucrare. Prelucrarea centralizată fără teleprelucrare asigură exercitarea unui bun control al prelucrării, o bună securitate privind accesul la date, dar influențează negativ timpul de răspuns al sistemului. Prelucrarea centralizată cu teleprelucrare este cea mai avantajoasă, dar necesită costuri mari de realizare.

III) Evaluarea fiecărei variante de realizare completează definirea funcționalității fiecărei variante cu o serie de informații referitoare la cheltuielile ocazionate și efectele economice scontate. Această comparare va permite beneficiarului să selecteze varianta cea mai eficientă. Evaluarea costurilor va urmări, atât cheltuielile prevăzute pentru proiectarea și realizarea sistemului informatic, cât și pe cele necesare exploatării curente a acestuia.

La prezentarea efectelor economice se vor avea în vedere, atât efectele economice cuantificabile, cât și efectele economice necuantificabile.

Evaluarea termenelor de realizare are în vedere precizarea intervalelor și termenelor de proiectare a noului sistem, precum și a momentelor de achiziționare a sistemelor de calcul necesare. De asemenea, această evaluare trebuie să precizeze termenele și intervalele de pregătire

profesională a personalului din posturile de lucru ce vor fi implicate în noul sistem, cât și a personalului din unitatea proprie de informatică.

Conducerea unității beneficiare evaluează și selectează varianta cea mai convenabilă prin luarea în considerare a tuturor variantelor, prin consultarea unor specialiști în analiza și realizarea de sisteme informatice sau (dacă este posibil) prin analiza unor soluții de informatizare alese de unități similare.

4.4. PROIECTAREA GENERALĂ A SISTEMELOR INFORMATICE

4.4.1. Obiective și activități specifice proiectării generale

Proiectarea generală are ca obiectiv elaborarea concepției logice a sistemului informatic, definirea acestuia din punct de vedere structural și funcțional. Aceasta presupune stabilirea componentelor sistemului informatic, a ieșirilor, a bazei informaționale de intrare, a documentelor pe care sunt consemnate datele de intrare, a legăturilor dintre ele și a funcționalității sistemului astfel încât toate elementele sale să formeze un întreg, să se articuleze ca un tot unitar.

Structura generală a sistemului informatic cuprinde un ansamblu de intrări, prelucrări și ieșiri definite în funcție de obiectivele noului sistem.

I. Intrările sistemului informatic cuprind :

- baza de date
- mișcările (tranzacțiile).

Baza de date ca noțiune generală reprezintă colecții de date operaționale, memorate și utilizate în comun de către componentele sistemului informatic.

Tranzacțiile sunt reprezentate de datele care reflectă modificări și devin componente ale bazei de date în urma clasificării, validării, înregistrării și prelucrării lor.

În cadrul unui sistem informatic există două categorii de tranzacții:

- tranzacții externe;
- tranzacții interne.

Tranzacțiile externe redau fenomenele și procesele economice, inclusiv modificările produse în activitatea agentului economic, ca de exemplu: aprovizionarea cu un material, livrarea unui produs, încasarea unei facturi, modificarea prețului unui produs etc.

Tranzacțiile interne apar în momentul obținerii unui rezultat, fiind destinate actualizării bazelor de date. De exemplu: totalul valorii facturilor primite de la un furnizor care actualizează la o anumită perioadă soldul furnizorului respectiv; totalul vânzărilor dintr-un produs într-o zi care actualizează stocul produsului respectiv etc.

Tranzacțiile externe provin din exteriorul sistemului de calcul și sunt furnizate prin proceduri de introducere a datelor din sistem, iar tranzacțiile interne sunt asigurate exclusiv în cadrul sistemului de calcul prin proceduri de actualizare și exploatare a colecțiilor de date create anterior.

II. Prelucrările sistemului informatic sunt asigurate prin execuția unui ansamblu de programe care se subdivid în: programe pentru actualizarea bazelor de date și programe pentru obținerea rezultatelor și a situațiilor de ieșire proiectate.

Deci sistemele informatice funcționează, la prima vedere, în două cicluri:

- actualizarea bazelor de date când se validează și se introduc tranzacțiile;
- obținerea situațiilor de ieșire solicitate.

Prelucrările sistemului informatic acționează asupra colecțiilor de date care sunt constituite într-un nucleu informațional ce conține mulțimea atributelor necesare obținerii informațiilor solicitate. Nucleul de informații urmează a fi structurat în entități informaționale al căror ansamblu delimitează baza informațională a sistemului informatic.

Baza informațională este formată din ansamblul entităților informaționale și atributelor

componente ale acestora ce descriu dinamica fenomenelor și proceselor economice la un agent economic pe o perioadă de timp.

Procedurile de creare, actualizare, exploatare, listare și colecțiile de date organizate în fișiere sau baze de date formează nucleul sistemului informatic.

III. Ieșirile sistemului informatic reprezintă rezultatul prelucrării colecțiilor de date și se vor concretiza în situații de ieșire solicitate de beneficiar sau în colecții de date: fișiere sau baze de date actualizate ce vor fi transmise altor sisteme sau aplicații informatice.

Proiectarea generală se derulează în mai multe faze:

- stabilirea obiectivelor pe care le va acoperi viitorul sistem;
- definitivarea conținutului informațional al ieșirilor conform cu obiectivele stabilite;
- determinarea bazei informaționale necesare obținerii ieșirilor proiectate;
- formalizarea atributelor de intrare și a elementelor structurale ale sistemului informatic prin codificare;
- proiectarea documentelor de intrare;
- proiectarea structurală și funcțională a noului sistem;
- elaborarea documentației proiectării generale.

Deoarece asigură definirea conceptuală a sistemului informatic, proiectarea generală este independentă față de soluțiile tehnice impuse de tipul calculatoarelor electronice, software-ului utilizat și modul de gestiune a bazelor de date.

Proiectarea generală se realizează în mai multe variante având în vedere aria de cuprindere a noului sistem, complexitatea obiectivelor, gradul de flexibilitate ce se cere noului sistem. În abordarea proiectării generale se poate pleca de la ieșiri la intrări, de la intrări la ieșiri sau se poate realiza o variantă mixtă.

În **varianta ieșiri - intrări** se pleacă de la obiectivele sistemului informatic și situațiile de ieșire unde se concretizează acestea.

Analizând modul de obținere a fiecărei informații se determină baza informațională de intrare, apoi se realizează celelalte faze ale proiectării.

Ordinea de parcurgere a fazelor este:

- stabilirea obiectivelor;
- proiectarea ieșirilor;
- proiectarea bazei informaționale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea structurală și funcțională;
- elaborarea documentației.

Varianta intrări - ieșiri poate avea la ieșire numai atributele existente în baza informațională de intrare sau indicatorii ce se pot calcula din atribute.

Această variantă se recomandă în cazul realizării unor sisteme informatice mari și mijloicii caracterizate prin complexitatea obiectivelor și a ieșirilor informaționale solicitate.

După stabilirea obiectivelor se inventariază documentele de intrare care circulă în sistemul agentului economic pentru determinarea bazei informaționale de intrare.

Ordinea de parcurgere a fazelor este:

- definirea obiectivelor;
- inventariere atributelor de intrare și a legăturilor dintre acestea pe baza documentelor de intrare folosite;
- proiectarea bazei informaționale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea ieșirilor sistemului informatic;
- proiectarea structurală și funcțională;
- elaborarea documentației.

Avantajul acestei variante este flexibilitatea bazei informaționale de intrare în condițiile

modificării cerințelor informaționale.

Dezavantajul este dat de baza informațională de intrare care este supradimensionată, ceea ce determină un timp mare de realizare, costuri ridicate și creșterea complexității prelucrărilor din sistemul informatic.

Varianta mixtă pornește prin stabilirea obiectivelor noului sistem și a unei variante inițiale a bazei informaționale de intrare, apoi avându-se în vedere posibilitățile de prelucrare se determină baza informațională de ieșire, iar în funcție de solicitările beneficiarului se proiectează ieșirile necesare în prezent și în perspectivă și pe baza lor se proiectează varianta finală a bazei informaționale de intrare.

Ordinea de parcurgere a fazelor este:

- definirea obiectivelor;
- proiectarea inițială a bazei informaționale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea ieșirilor sistemului informatic;
- reproiectarea bazei informaționale de intrare;
- proiectarea structurală și funcțională;
- elaborarea documentației.

4.4.2. Proiectarea ieșirilor sistemului informatic

Realizarea practică a obiectivelor oricărui sistem informatic se concretizează prin satisfacerea cerințelor informaționale ale sistemului de conducere din organismul economic pe care acesta este grefat, respectiv prin furnizarea ieșirilor.

Ieșirile sistemului informatic pot fi privite din trei puncte de vedere:

- structural;
- funcțional;
- tipologic.

Din punct de vedere structural ieșirile sistemului informatic reprezintă a treia componentă din triada ce caracterizează structura oricărui tip de sistem, respectiv INTRĂRI - PRELUCRĂRI - IEȘIRI.

Din punct de vedere funcțional ieșirile sistemului informatic concretizează obiectivele generale și specifice ale sistemului.

Din punct de vedere tipologic ieșirile sistemului informatic pot fi realizate sub formă de:

- liste (rapoarte sau situații) de ieșire care cuprind indicatori ai proceselor, stărilor și rezultatelor activității economice;
- grafice, care redau într-o formă sinoptică starea și evoluția indicatorilor economici;
- ieșiri către alte sisteme informatice sub forma unor colecții organizate de date (fișiere sau baze de date); acestea pot fi transmise indirect (off-line) prin intermediul suporturilor tehnice de date (disc magnetic, bandă magnetică, etc.) sau direct (on-line) prin mijloace de teletransmisie.

Într-un sistem informatic economic listele ce grupează indicatori economici constituie ponderea situațiilor de ieșire, dar se constată, în ultimul timp, o creștere a ponderii ieșirilor sub formă grafică, proces ce se va amplifica pe măsura implicării crescânde a sistemelor informatice în procesul decizional.

În proiectarea situațiilor de ieșire se va ține seama de următoarele aspecte:

- a) destinația situației și momentul când se estimează că se poate obține, nivelul de detaliere a informației și anume: informații mai analitice la nivelurile inferioare ale conducerii și informații de sinteză la nivelul superior;
- b) existența în raport a tuturor informațiilor necesare beneficiarului pe segmentul stabilit;
- c) aceeași informație să nu apară în mod inutil în mai multe situații, mai ales dacă este vorba de o

informație ce se obține în urma unor prelucrări mai complexe.

O situație de ieșire trebuie să îndeplinească, în general, următoarele condiții:

- să se refere la unul din obiectivele sistemului;
- să acopere integral domeniul la care se referă;
- să fie cât mai concisă, să elimine detaliile ce nu sunt necesare;
- să fie prezentată într-o formă simplificată și inteligibilă;
- să apară la timp;
- informațiile pe care le conține să fie corelate cu informațiile din celelalte situații.

Conținutul și forma situațiilor de ieșire se stabilesc de către colectivul de proiectare împreună cu beneficiarul sistemului care va viza nominal macheta fiecărei situații de ieșire.

Situațiile de ieșire dintr-un sistem informațional economic pot fi clasificate după următoarele criterii:

- a) specificul funcțiilor generale ale agentului economic;
- b) destinație;
- c) gradul de sintetizare a indicatorilor;
- d) momentul generării;
- e) intervalul de referință;
- f) modul de obținere.

a) După specificul funcțiilor generale ale agentului economic urmărit în funcție de natura activității de bază situațiile de ieșire se pot grupa în grupe omogene corespunzătoare funcțiilor agentului economic:

- situații de ieșire pentru activitatea de producție;
- situații de ieșire pentru activitatea comercială;
- situații de ieșire pentru activitatea financiar-contabilă;
- situații de ieșire pentru activitatea de personal.

b) După destinație situațiile de ieșire pot fi:

- situații de ieșire destinate altor sisteme;
- situații de ieșire destinate sistemului propriu.

Situațiile de ieșire destinate altor sisteme (agenți economici sau organisme de stat) au ca scop furnizarea de informații specifice cooperării în realizarea activităților proprii și de informare a organismelor de stat. Acestea sunt stabilite prin acte normative sau prin înțelegeri între unități, motiv pentru care structura lor este obligatorie pentru proiectant. În această categorie se includ rapoartele statistice ale principalilor indicatori economico-financiar, dările de seamă, situațiile operative între unități referitoare la stadiul relațiilor contractuale, etc. Tot aici se includ și documentele primare generate prin prelucrare automată a datelor (factura fiscală, avizul de însoțire a mărfii etc.) care circulă și în exteriorul sistemului propriu.

Situațiile de ieșire destinate sistemului propriu al agentului economic cuprind informații necesare atât compartimentelor funcționale, cât și organelor de conducere ale unități. De exemplu: Situația realizării producției, Situația livrărilor pe beneficiari, Situația facturilor emise și încasate, etc.

c) Din punct de vedere al gradului de sintetizare a indicatorilor avem:

- situații de ieșire ce conțin informații cu caracter sintetic;
- situații de ieșire ce conțin informații cu caracter analitic.

Situațiile de ieșire ce conțin informații cu caracter sintetic servesc pentru conducerea de ansamblu a agentului economic și cuprind indicatori globali de urmărire și control cum ar fi : profitul, cifra de afaceri, productivitatea muncii, consumurile, producția marfă etc.

Situații de ieșire ce conțin informații cu caracter analitic descriu în detaliu elementele patrimoniale ale agentului economic, operațiile economice legate de acestea și sunt destinate compartimentelor funcționale pentru realizare unor lucrări curente. De exemplu: Fișa de urmărire a aprovizionării, Situația materialelor fără mișcare, Graficul de livrare, Situația realizării produc în ziua de .. , etc.

d) Momentul generării asigură obținerea de situații cu caracter:

- operativ;
- periodic;
- aleator.

Situațiile de ieșire cu caracter operativ sunt elaborate la perioade mai mici (schimb, zi, semidecadă, decadă, etc.) și conțin informații operative cu un grad redus de prelucrare necesare conducerii prin excepție. De exemplu: Situația realizării producției în ziua de .., Analiza realizării planului de desfacere în ziua de ..., etc.

Situațiile de ieșire cu caracter periodic se elaborează la intervale mai mari de timp (lunar, trimestrial, anual) și conțin informații sintetice ce caracterizează evoluția fenomenelor și proceselor economice. Sunt destinate anumitor factori de decizie, compartimentelor funcționale sau altor sisteme și se caracterizează prin termene fixe de elaborare. De exemplu: Situația facturilor emise și încasate pe luna ... , Balanța analitică a materialelor pe luna... , Balanța sintetică pe luna ..., etc.

Situațiile de ieșire cu caracter aleator sunt obținute cu frecvențe și la termene nedeterminate și conțin informații sintetice sau analitice ce reflectă aspecte pozitive sau negative din activitatea agentului economic, fiind destinate, de regulă, conducerii de pe treptele superioare (director, consiliu de administrație, etc.) De exemplu: Situația stocurilor ce nu au mișcare de "n" zile, Situația facturilor neîncasate de "n" zile, etc.

e) După intervalul de referință situațiile de ieșire se pot structura în trei categorii:

- de stare;
- statistice;
- previzionale.

Situațiile de ieșire cu caracter de stare conțin informații ce evidențiază nivelul și disponibilul de resurse la un moment dat, reflectând elemente de patrimoniu ale agentului economic cum ar fi: stocuri de materiale și produse, facturi în sold, nivelul veniturilor, etc.

Situațiile de ieșire cu caracter statistic grupează informații pe mai multe perioade de gestiune pentru a stabili tendințele evolutive ale fenomenelor și proceselor economice în vederea formulării unor aprecieri de ansamblu. Din această categorie fac parte situațiile transmise Comisiei Naționale de Statistică, organelor Ministerului Finanțelor sau altor organisme publice. Situațiile de ieșire cu caracter previzional redau informații ce reflectă tendințele fenomenelor și proceselor economice în scopul stabilirii obiectivelor viitoare ale agentului economic.

f) După modul de obținere situațiile de ieșire sunt:

- situații de ieșire obținute la imprimantă;
- situații de ieșire obținute la videoterminal.

Situații de ieșire obținute la imprimantă se folosesc de regulă în cazul în care cadrul legislativ prevede utilizarea lor ca documente justificative ce se vor arhiva după consultare. De exemplu: Statele de salarii, Jurnalul privind vânzările, Situația centralizatoare a intrărilor de materiale, etc.

Situații de ieșire obținute la videoterminal sunt de regulă situații de urmărire operativă a unor indicatori economici și se folosesc în toate cazurile în care cadrul legislativ nu prevede utilizarea lor ca documente justificative.

În definirea conținutului și modului de obținere a situațiilor de ieșire proiectantul va ține seama de cerințele beneficiarului dar va avea în vedere și performanțele și limitele sistemelor electronice de calcul și ale mediului de programare cu care lucrează.

La definitivarea fiecărei situații de ieșire trebuie să se precizeze următoarele elemente:

- antetul situației, care precizează date de identificare a unității (subunității) beneficiare, data și eventual codul situației;
- titlul situației, care redă într-o formă sintetică conținutul informațional al acesteia și perioada de referință;
- capul de tabel, respectiv prezentarea indicatorilor din fiecare coloană urmărindu-se gruparea logică a acestora : indicatori informativi (de exemplu cod, denumire),

- indicators de grupare (de exemplu unitate de măsură, preț) și indicators cantitativ-valorici;
- natura și lungimea maximă a fiecărui indicator;
- benzile de total;
- numărul de exemplare;
- destinația fiecărui exemplar;
- periodicitatea și termenele de obținere;
- dispozitivul periferic de obținere.

4.4.3. Proiectarea bazei informaționale de intrare

Proiectarea bazei informaționale de intrare înseamnă determinarea completă și corectă a mulțimii atributelor de intrare necesare și suficiente pentru obținerea ieșirilor sistemului informatic și gruparea acestora într-un ansamblu omogen de entități în vederea construirii în proiectarea de detaliu a fișierelor sau a bazelor de date.

Proiectarea bazei informaționale de intrare are două subfaze:

- a) determinarea conținutului bazei informaționale de intrare și a algoritmilor folosiți;
- b) structurarea bazei informaționale în entități.

a) Determinarea conținutului bazei informaționale de intrare.

Pentru a stabili necesarul atributelor de intrare trebuie examinat modul în care are loc obținerea informațiilor cuprinse în situațiile de ieșire proiectate sau în documentele inventariate în sistem (în funcție de varianta de lucru aleasă în proiectarea generală).

Din acest punct de vedere se disting două categorii de informații:

- informații ce se obțin în urma aplicării unui algoritm de calcul (de exemplu: stocul unui material, soldul unui cont, numărul de zile restanță la plata unei facturi, etc.)
- informații obținute prin simpla transferare a unor valori similare aflate la intrare (de exemplu: codul unui produs, denumirea unui produs, numele unei persoane, etc.)

Informațiile din prima categorie trebuie descompuse pe baza algoritmului de calcul în operanzi primari ai acestora. Apoi aceștia împreună cu informațiile preluate (transferate) din a doua categorie reprezintă elementele necesare obținerii informațiilor solicitate.

Deci, ansamblul atributelor de intrare este reprezentat de totalitatea informațiilor obținute prin preluare și a operanzilor primari după reducerea elementelor comune.

Se poate defini următorul procedeu de determinare a bazei informaționale de intrare:

- se inventariază câmp cu câmp informațiile cuprinse în situațiile de ieșire proiectate;
- dacă informațiile rezultă dintr-un calcul se identifică algoritmul, apoi operanzii primari se introduc în baza informațională de intrare dacă nu au fost deja incluși;
- dacă informația este transferată se include în baza informațională de intrare dacă nu a fost inclusă deja.

Consistența și relevanța bazei informaționale de intrare sunt dependente de acuratețea algoritmilor în corespondență cu specificul sistemului obiect.

Ansamblul atributelor de intrare va fi apoi analizat și structurat în entități, iar algoritmi de calcul vor fi descriși în procedurile de exploatare. Algoritmi determinați se pot exprima ca expresii algebrice, liste de acțiuni condiționale, tabele de decizii.

b) Structurarea bazei informaționale de intrare în entități.

Cea de a doua subfază presupune separarea ansamblului atributelor de intrare în grupe omogene și stabilirea legăturilor dintre acestea. Pentru aceasta se rezolvă următoarele probleme:

- se analizează baza informațională de intrare determinată anterior;
- se grupează attributele în entități.

Analiza bazei informaționale de intrare are scopul de a determina modul de participare a datelor la procesul de prelucrare automată.

Pentru a efectua această analiză se au în vedere elementele funcționale indivizibile ale

unui sistem informatic, respectiv entitățile.

Entitatea poate fi privită ca un aspect al existenței delimitat ca întindere, conținut și sens și desemnează un element conceptual sau material indivizibil din punct de vedere funcțional. Ea este deci o componentă structural-funcțională elementară a sistemului care are o existență bine determinată și reprezintă o individualitate. De exemplu: entitatea personal, entitatea materiale, entitatea costuri, etc.

Entitatea este caracterizată printr-o mulțime de atribute preluate din nucleul informațional în urma aplicării unor criterii de structurare. De exemplu entitatea personal are ca atribute: numele persoanei, prenumele persoanei, sexul, data nașterii, etc.

Pentru a realiza analiza corectă a bazei informaționale de intrare este necesară distincția clară între atribut și valoarea pe care o poate avea. De exemplu NUME este un atribut (un nume de dată, o caracteristică) a entității PERSONAL iar Popa, Sandu, Radu, etc. sunt valori pe care le ia atributul respectiv. Din perspectiva proiectării și realizării unui sistem informatic interesează identificarea atributului și nu a valorilor concrete pe care acesta le poate lua.

Valoarea informațională a unei entități cuprinde trei aspecte:

- caracteristici determinante specifice care identifică entitatea și o individualizează în ansamblul sistemului;
- nivelul sau starea entității la un moment dat;
- transformările sau schimbările care intervin datorită funcțiilor sistemului și determină trecerea de la o stare la alta.

Poziția unui atribut în raport cu o entitate îi conferă acestuia un mod specific de comportare și de aici rezultă trei feluri de atribute: cu caracter permanent, variabil și de stare.

Atributele permanente rămân neschimbate o perioadă mai mare de timp, se preiau în sistem o singură dată și se actualizează la intervale mai mari de timp. Ele intră în componența entităților permanente. De exemplu: marca, nume, sex, etc. sunt atribute permanente ale entității de personal.

Atributele variabile se caracterizează prin aceea că valoarea lor variază de la o prelucrare la alta reflectând fenomene și procese economice ce s-au desfășurat de la prelucrarea precedentă. Ele sunt introduse în sistem la fiecare prelucrare și intră în componența entităților variabile. De exemplu: timpul lucrat, producția realizată de o persoană, indicele individual de acord, etc. sunt atribute variabile ale entității de personal.

Atributele de stare caracterizează nivelul unui element la un moment dat și reprezintă rezultatul unei prelucrări. Ele sunt stocate în baza informațională de unde se preiau periodic, se prelucrează și se înregistrează la noua valoare. Ele nu reflectă fenomene și procese economice ce au avut loc între două prelucrări, deci sunt diferite de atributele variabile, dar nu au nici o valoare constantă, ci ea se schimbă la fiecare prelucrare și, deci, nu sunt nici atribute permanente. Ele se introduc în entități permanente, iar când sunt mai numeroase se constituie în entități de stare. De exemplu: vârsta, numărul de copii, suma câștigurilor de la începutul anului, etc. sunt atribute de stare ale entității de personal.

Pe baza acestei abordări poate fi definit modelul conceptual de prelucrare pe calculator la fiecare ciclu astfel:

- se actualizează atributele permanente cu eventualele modificări intervenite;
- se introduc în sistem valorile atributelor variabile prin preluarea din documente justificative a datelor ce reflectă operații economice efectuate de la prelucrarea anterioară;
- se actualizează atributele de stare pe baza valorii inițiale și a mișcărilor reflectate de atributele variabile, se calculează starea finală pentru fiecare atribut de stare;
- se asociază și se prelucrează atributele permanente, variabile și de stare pentru obținerea situațiilor de ieșire proiectate.

Gruparea atributelor în entități are la bază analiza efectuată anterior și constă în separarea ansamblului unic de date în grupuri omogene în funcție de tipul atributelor, de aici rezultă că și entitățile sunt permanente, variabile și de stare.

Relațiile dintre entități pot fi:

- atributele din entități permanente se actualizează pe baza tranzacțiilor externe ce reflectă modificări în structura și componența acestora;
- entitățile variabile se creează la fiecare prelucrare pe baza tranzacțiilor externe ce reflectă operațiile economice;
- entitățile de stare se actualizează pe baza tranzacțiilor interne.

Gruparea datelor în entități permanente se face în funcție de următoarele criterii:

- atribute de identificare;
- frecvența de utilizare și actualizare;
- accesul la date.

Primul criteriu impune gruparea în aceeași entitate a tuturor atributelor ce au un element de identificare comun. De exemplu salariul, numărul de copii, data nașterii, adresa se pot grupa într-o singură entitate având comun elementul de identificare marca sau numele.

Al doilea criteriu determină gruparea în aceeași entitate a atributelor care au aceeași frecvență de utilizare și actualizare. De exemplu sporul de noapte, sporul șantier, sporul pentru condiții deosebite se pot grupa într-o altă entitate decât cea anterioară (deși se referă la aceeași persoană) datorită criteriului precizat.

Al treilea criteriu se referă la gruparea în aceeași entitate a atributelor pentru care sunt formulate cerințe deosebite legate de consultarea bazelor de date și are importanță deosebită în sistemele interactive la care este necesar să se asigure un timp de răspuns cât mai redus.

Gruparea datelor în entități variabile se face avându-se în vedere posibilitatea grupării sub un identificator comun și în funcție de omogenitatea datelor din punct de vedere al locului și momentului preluării lor. Aceasta impune gruparea în aceeași entitate a tuturor atributelor ce se regăsesc pe același document de mișcare.

În cazul în care numărul de documente este relativ mare, iar ponderea lor este inegală în sistem se poate apela la gruparea atributelor în funcție de operațiile economice pe care le reflectă. De exemplu ieșirile de produse se reflectă în mai multe documente: factura fiscală, avizul de însoțire a mărfii, lista de inventariere, etc. În măsura în care ieșirile prin vânzare au ponderea cea mai mare se poate apela la gruparea atributelor ce reflectă toate ieșirile într-o singură entitate, codificându-se tipurile de ieșiri.

Gruparea atributelor în entități de stare se face în funcție de aceleași criterii ca și cele avute în vedere la entitățile permanente. De regulă, numărul atributelor de stare este mai mic și soluția cea mai avantajoasă este aceea a grupării lor în entitățile permanente la care se referă, obținându-se astfel entități cu un grad mai ridicat de integrare. De exemplu atributele stoc și data ultimei mișcări pot fi înregistrate în entitatea permanentă produs care va oferi astfel o imagine mai completă asupra unui produs la un moment dat.

Constituirea entităților în cadrul bazei informaționale de intrare poate fi privită ca o operație de separare a ansamblului unic de date în mai multe grupe omogene. Aceste grupe ar trebui, la prima vedere, să constituie submulțimi disjuncte pentru a evita preluările repetate. Dar, în cursul prelucrării, va fi necesară reconstituirea parțială a ansamblului inițial de atribute, respectiv cuplarea a două sau mai multe entități. Pentru aceasta entitățile ce se vor cupla trebuie să aibă atribute comune pentru a permite realizarea asocierilor necesare.

Atributele prezente simultan în două sau mai multe entități se numesc atribute de legătură. Ansamblul atributelor de legătură dintre entități se determină pe baza asocierilor pe care le impun informațiile ce vor trebui obținute în situațiile de ieșire și algoritmi de calcul.

Trebuie menționat că nu toate atributele ce se pot găsi simultan în mai multe baze de date pot constitui atribute de legătură. Pentru aceasta atributul respectiv trebuie să constituie identificator pentru cel puțin una din entitățile asociate. De exemplu pentru obținerea unei situații de ieșire utilizăm entitățile cu structura de mai jos:

- entitatea produs cu atributele: cod produs, denumire, unitate de măsură, preț;
- entitatea mișcări cu atributele: cod produs, preț, cantitate.

Atributul preț, deși este prezent în ambele entități, nu poate servi drept atribut de legătură

deoarece nu identifică în mod unic nici una din entitățile ce cuplează. De aceea se va folosi ca atribut de legătură codul produsului.

4.4.4. Proiectarea codurilor

O cerință de bază a sistemelor informatice este aceea ca elementele din sistem să fie riguros definite, ordonate și clasificate prin codificare.

Codificarea reprezintă o activitate cu implicații mari în prelucrarea automată a datelor regăsindu-se sub două aspecte :

- pe de o parte, ca o codificare internă a datelor în calculatorul electronic;
- pe de altă parte, ca o codificare externă, ceea ce permite introducerea în prelucrare a unor date cât mai împachetate, mai formalizate.

Prin codificare înțelegem generarea unor grupuri de simboluri, denumite coduri și repartizarea acestora valorilor concrete pe care le iau elementele diferitelor mulțimi (în cazul unui sistem informatic se codifică atât atribute din baza de date cât și elementele structurale și funcționale ale sistemului)

Rezultatul activității de codificare este concretizat într-un sistem de coduri.

Prin cod se înțelege o combinație de simboluri asociate unui ansamblu de date. Între mulțimea simbolurilor ce urmează a fi codificate și mulțimea codurilor asociate se stabilește o corespondență biunivocă.

Totalitatea combinațiilor distincte, posibile de realizat din simbolurile ce compun un cod reprezintă capacitatea unui sistem de coduri.

Un cod trebuie să aibă o capacitate acoperitoare pentru toate situațiile posibile cu condiția păstrării unicității codului.

Capacitate unui sistem de coduri numeric se determină cu formula:

$$C = b^n - 1$$

unde : C = capacitatea codului

b = baza sistemului de numerație utilizat

n = numărul de poziții numerice din cadrul codului

Capacitatea unui sistem de coduri alfanumeric se determină cu formula:

$$C = 24^a * b^n$$

unde : C = capacitatea codului

a = numărul de poziții alfabetice din cadrul codului

b = baza sistemului de numerație utilizat

n = numărul de poziții numerice din cadrul codului

Numărul de simboluri elementare dintr-un cod poartă denumirea de lungime a secvenței de cod (lungimea codului).

Lungimea codului trebuie să fie minimă pentru a reduce timpul de transmisie și preluare a datelor pe suportii tehnici de date, pentru a reduce timpul de prelucrare etc.

În același timp, lungimea codului trebuie să-i asigure și o capacitate corespunzătoare.

Aceasta se poate estima cu relația: $L \geq \log_K N$

unde: L = lungimea codului

N = numărul de elemente ce se codifică

K = numărul de simboluri utilizate în construirea codului

Forma finală a codului cu precizarea clară a numărului de poziții utilizate, natura acestora, cifra de control și algoritmul de calcul al acesteia reprezintă formatul codului.

În proiectarea unui sistem de coduri trebuie să avem în vedere două aspecte importante și anume:

- influența tipului și structurii codului asupra performanțelor operațiilor de prelucrare automată a datelor din sistemul informatic;

- implicațiile utilizării codurilor în operațiile de culegere și preluare a datelor și de interpretare a rezultatelor finale de către utilizatorii neinformaticieni.

Având în vedere aceste considerente, se impune ca în proiectarea unui sistem de coduri să se respecte o serie de cerințe:

1. **Unicitate.** Fiecărui element din mulțimea codificată i se atribuie un cod unic, această cerință trebuind să fie asigurată la nivelul întregului sistem informatic.
2. **Stabilitate.** Caracteristica codificată trebuie să rămână neschimbată o perioadă cât mai mare de timp.
3. **Elasticitate.** Să permită inserări și extensii ale nomenclatoarelor de coduri în vederea includerii de noi elemente.
4. **Conciziune.** Să se utilizeze un număr cât mai mic de simboluri în construirea unui cod.
5. **Claritate.** Să permită realizarea cu ușurință a operațiunilor de codificare-decodificare.
6. **Semnificație.** Să permită, pe cât posibil, sugerarea caracteristicilor codificate și să genereze interes pentru a facilita utilizarea codurilor.
7. Codul să fie **operațional.** Să permită prelucrarea automată a datelor și efectuarea cu ușurință a operațiilor de sortare, indexare, scindare, interclasare etc.

Un sistem de coduri are următoarele funcții:

a) **Funcția de caracterizare**, care asigură exprimarea într-o formă concisă, unică și stabilă în timp, a conținutului semantic al fiecărui element codificat prin intermediul codului asociat acestuia. În mod concret funcția de caracterizare permite utilizarea cu prioritate a codului în locul denumirii integrale a elementului codificat.

b) **Funcția de identificare**, care oferă posibilitatea regăsirii mai rapide a elementelor prin intermediul codurilor asociate lor decât prin folosirea completă a semanticii acestora. Această funcție creează posibilitatea ulterioară de selectare a anumitor caracteristici prin intermediul cărora se vor identifica în mod unic valori folosind conceptul de cheie de acces.

c) **Funcția de control**, care presupune existența unei chei de control (formată din unul sau mai multe caractere) pe baza căreia folosind metode și algoritmi specifici să se poată verifica integral corectitudinea simbolurilor care intră în structura codului. De regulă, această cheie de control se plasează în ultima parte a codului și este separată de acesta printr-o linie de unire.

d) **Funcția de manipulare** a elementelor codificate, care facilitează introducerea în memorie a acestora, reducerea timpului de prelucrare, inclusiv ușurința folosirii codului de către personalul din compartimentele funcționale implicate în funcționarea sistemului informatic.

Tipuri de coduri. Diversitatea și complexitatea colecțiilor de date, specificul operațiilor de regăsire, sortare, indexare etc. au condus la apariția unei palete variate de coduri, ce se pot grupa după mai multe criterii .

I). După natura caracterelor utilizate:

1. *Coduri numerice*, în care simbolurile utilizate în construirea codului sunt cifrele de la 0 la 9. De exemplu codificând secțiile din facultate: 11 pentru Management; 12 pentru Contabilitate; 13 pentru Informatică etc.

2. *Coduri alfabetice*, în care simbolurile utilizate în construirea codului sunt literele alfabetului. De exemplu : MIZ pentru Management, cursuri de zi; CIS pentru Cibernetică, informatică și statistică etc.

3. *Coduri alfanumerice*, în care simbolurile utilizate sunt cifrele, literele și toate caracterele speciale cuprinse în codul ASCII.

II. După lungimea secvenței de cod există:

1. *Coduri cu lungime fixă*, în care toate elementele unei mulțimi sunt codificate cu același număr de caractere.

2. *Coduri cu lungime variabilă*, în care lungimea secvenței de cod poate fi diferită pentru elemente diferite din aceeași mulțime.

III. După semnificația codului sunt:

1. *Coduri semnificative.* Aceste coduri semnifică conținutul informațional al elementelor pe care le reprezintă. De exemplu: lect - semnifică valoarea lector, conf - semnifică valoarea conferențiar, stud - semnifică student etc.

2. *Coduri ne semnificative.* Aceste coduri nu semnifică conținutul informațional al elementelor pe care le reprezintă. Ele sunt combinații de simboluri după anumite criterii și se atribuie elementelor fără a avea o semnificație. De exemplu: 01 - pentru lector, 02 - pentru conferențiar etc.

IV. După modul de întocmire a nomenclatoarelor de coduri există:

1. *Coduri atribuite manual;*
2. *Coduri atribuite automat.*

V. Din punct de vedere al utilizatorului se pot folosi:

1. *Coduri accesibile utilizatorului;*
2. *Coduri ascunse utilizatorului.*

VI. După structura codului există:

1. *Coduri elementare;*
2. *Coduri complexe.*

Codurile elementare au rolul de a identifica un element din cadrul unei singure mulțimi de elemente. Ele pot fi:

- coduri secvențiale,
- coduri secvențiale cu formare de grupe,
- coduri abreviate.

Codurile secvențiale se construiesc prin atribuirea în ordine crescătoare a unor simboluri numerice sau alfanumerice elementelor din mulțimea ce urmează a fi codificată, pe măsură ce aceste elemente apar în sistem. Ele pot avea lungimă fixă sau variabilă. În cazul codurilor cu lungime fixă se adaugă zerouri ne semnificative în fața șirurilor de caractere atribuite.

Codurile secvențiale au avantajul conciziunii, dar nu sunt elastice și nu pot fi utilizate eficient în gruparea datelor.

Codurile secvențiale cu formare de grupe. Reprezintă o dezvoltare a codurilor secvențiale, în sensul că se prevede rezervarea unor grupe de coduri pentru grupe de elemente, iar în cadrul grupelor elementele sunt codificate secvențial.

Separarea mulțimii elementelor de codificat pe grupe sau subgrupe se face pe baza unor caracteristici comune tuturor elementelor ce aparțin grupei sau subgrupei respective.

Acest tip de coduri este elastic, servind și necesităților de grupare și ordonare a datelor.

Coduri abreviate. Sunt coduri alfabetice, de lungime fixă sau variabilă care se constituie prin prescurtare sau prin preluarea unor inițiale.

Cele obținute prin prescurtare poartă denumirea de *mnemonice*.

Cele obținute prin preluarea unor inițiale poartă denumirea de *acronime*.

Codurile complexe. Se folosesc pentru elementele ce pot să aparțină mai multor mulțimi distincte care pot fi folosite în comun pentru prelucrări viitoare. Aceste coduri sunt astfel construite încât să reflecte apartenența multiplă. El pot fi:

- coduri ierarhizate,
- coduri juxtapuse,
- coduri combinate.

Codurile ierarhizate se folosesc atunci când între caracteristicile ce urmează a fi simbolizate (codificate) sunt stabilite relații de subordonare.

Codurile juxtapuse (partajate) se construiesc prin concatenarea codurilor atribuite caracteristicilor individuale cu semnificații distincte.

Codurile combinate surprind în structura lor relațiile logice existente între diferite caracteristici ale elementelor de codificat. Ele pot fi matriciale sau binare.

Codurile matriciale se folosesc în cazul în care elementele de codificat pot fi caracterizate în funcție de două însușiri, codul atribuit reprezentându-le cumulativ pe amândouă.

Codurile binare se folosesc pentru a reflecta în mod sugestiv și simplu prezența sau absența mai multor însușiri ce caracterizează simultan elementele unei mulțimi de date. În acest sens, însușirile respective se codifică prin puteri succesive ale cifrei doi, fiecare poziție putând lua valoarea 0 sau 1 în funcție de prezența sau absența însușirii respective.

VII. După modul de preluare în sistemul de calcul, codurile pot fi:

1. *Coduri preluate manual* (prin tastare de la terminal);
2. *Coduri preluate automat*. Ele sunt introduse în sistem prin utilizarea unor echipamente periferice speciale precum: echipamente pentru citirea documentelor scrise cu cerneală magnetică, scannere, camere video și alte echipamente multimedia.

VIII. După modul în care tratează erorile provenite în urma utilizării unui sistem de coduri întâlnim:

1. *Coduri care nu detectează erorile de manipulare;*
2. *Coduri autodetectoare de erori;*
3. *Coduri autocorectoare de erori.*

Activitățile parcurse în elaborarea unui sistem de coduri sunt:

1. analiza elementelor ce urmează a fi codificate;
2. precizarea și uniformizarea terminologiilor;
3. stabilirea caracteristicilor și a relațiilor dintre elementele de codificat;
4. alegerea tipurilor de coduri;
5. estimarea capacității, lungimii și formatului codurilor;
6. determinarea cifrei de control corespunzătoare fiecărui cod și asocierea ei codului respectiv;
7. atribuirea codurilor elementelor de codificat, respectiv crearea nomenclatoarelor de coduri;
8. întreținerea nomenclatoarelor de coduri.

O problemă deosebită care se ridică în legătură cu codificarea datelor este cea a verificării corectitudinii codului în procesele de culegere, transmitere și prelucrare a datelor.

Cu ocazia manipulării codului, apar posibilități ca acesta să fie modificat, erorile astfel generate având consecințe negative asupra corectitudinii informațiilor obținute și asupra funcționării sistemului informatic în ansamblu.

Pe parcursul exploatării sistemului informatic pot să apară erori datorate:

- transpunerii greșite pe documentul justificativ sau pe elementul pe care codul îl identifică;
- preluării incorecte (de pe documentele justificative de exemplu) în sistemul de calcul.

S e pune deci problema de a utiliza metode ce pot prevenii sau depista erorile în codificare și în utilizarea codurilor. În acest sens, o tehnică des utilizată este cea a cheii (cifrei) de control.

4.5. PROIECTAREA DE DETALIU

4.5.1. Caracteristicile generale ale proiectării de detaliu

Obiectivul fundamental al proiectării de detaliu constă în realizarea cerințelor funcționale definite în proiectarea generală, prin transformarea modelului conceptual într-un model tehnic, operațional. În acest scop se alege soluția optimă de gestiune a datelor, bazată pe principiul utilizării fișierelor sau a bazelor de date și se proiectează structurile de date, inclusiv prelucrările specifice la nivelul subsistemelor (sau al sistemului informatic în ansamblu) și prelucrările specifice de la nivelul procedurilor.

Proiectarea de detaliu are un caracter preponderent tehnic, iar activitățile desfășurate sunt influențate direct de soluția de gestiune a datelor aleasă, de caracteristicile sistemului electronic de calcul, de sistemul de operare, precum și de mediul de programare ce va fi utilizat.

În realizarea practică a proiectării de detaliu a sistemului informatic se parcurg următoarele faze: proiectarea fișierelor; stabilirea ordinii de prelucrare a fișierelor; determinarea procedurilor; proiectarea procedurilor.

4.5.2. Proiectarea fișierelor

Urmărește determinarea celei mai adecvate structuri de date pentru entitățile bazei informaționale anterior stabilite, în vederea optimizării obținerii ieșirilor proiectate. Pentru aceasta trebuie realizate următoarele activități:

- definitivarea dicționarului de date și a clasificării fișierelor de bază,
- proiectarea structurii înregistrărilor,
- alegerea modului de organizare a fișierelor de bază și a suporturilor de memorie externă,
- definirea măsurilor de protecție și securitate.

4.5.3. Stabilirea ordinii de prelucrare a fișierelor de bază

Ordinea de prelucrare a fișierelor de bază este determinată de restricțiile de integritate dintre fișiere, determinate ca reguli sau condiții formulate în scopul menținerii coerenței datelor în timpul operațiunilor de prelucrare.

Aceste restricții sunt evidențiate prin subordonarea dintre fișiere, care impun ca valoarea unuia sau mai multor câmpuri dintr-un fișier să se regăsească în calitate de cheie primară sau externă pentru alt fișier.

Pentru a exemplifica, să considerăm fișierul PRODUS (un nomenclator ce conține date permanente și de stare) și MIȘCĂRI (un fișier cu date variabile ce conține intrările și ieșirile de produse). Fiecare produs ce se găsește în fișierul MIȘCĂRI trebuie să aibă un corespondent în fișierul PRODUS, câmpul COD_PROD fiind în același timp cheie primară pentru fișierul PRODUS și cheie externă pentru fișierul MIȘCĂRI.

Restricțiile de integritate dintre fișiere determină ordinea de prelucrare a lor, astfel încât crearea și actualizarea unui fișier să se facă numai după ce s-a efectuat actualizarea tuturor fișierelor de care acesta depinde. Regula este că se actualizează în primul rând fișierele cu date permanente (nomenclatoarele) și apoi fișierele variabile. În acest context fișierul PRODUS trebuie actualizat înaintea prelucrării fișierului MIȘCĂRI, pentru a putea reflecta și mișcările referitoare la produsele noi.

Restricțiile de integritate dintre fișiere pot fi privite în sens static și în sens dinamic.

În sens static acestea reflectă, după cum am văzut, modul de subordonare a prelucrărilor unui fișier în raport cu altul, ceea ce condiționează în mod direct ordinea de prelucrare.

Restricțiile de integritate dintre fișiere, abordate în sens dinamic exprimă reguli sau corelații existente înainte sau pe parcursul prelucrării acestora și sunt, în general, dictate de particularitățile sistemului obiect. De exemplu, în cadrul operațiilor privind mișcările de produse vor fi prelucrate mai întâi intrările și apoi ieșirile, pentru a evita apariția de stocuri intermediare negative (anomalie din punct de vedere economic).

Deci într-o aplicație informatică economică ordinea de apelare a procedurilor trebuie să fie următoarea: proceduri de actualizare a fișierelor permanente; proceduri de actualizare a fișierelor variabile; proceduri de actualizare a datelor (fișierelor) de stare; proceduri de obținere a ieșirilor.

4.5.4. Determinarea procedurilor

Într-un sistem informatic economic procedura poate fi definită ca o secvență de operații repetitive executate fără întreruperi externe de la același post de lucru, care transformă un anumit ansamblu de intrări în anumite ieșiri, după reguli bine definite.

Pentru fiecare procedură va trebui să specificăm următoarele elemente: funcția procedurii; intrările în procedură; ieșirile din procedură; logica internă de prelucrare; interfețele cu alte proceduri.

Funcția procedurii indică natura prelucrărilor sale asupra fișierelor, existând mai multe tipuri: proceduri pentru dirijarea altor proceduri (așa numitele proceduri de comandă); proceduri pentru introducerea și validarea datelor; proceduri pentru actualizarea fișierelor; proceduri pentru exploatarea fișierelor și obținerea ieșirilor; proceduri pentru reorganizarea fișierelor de bază; proceduri pentru protecția și securitatea fișierelor de bază.

Intrările în proceduri sunt constituite din parametri de apel și date de intrare din fișierele de bază. Fiecare tip de intrare trebuie descris, în documentație, în forma standard, respectiv cu specificatorul complet de fișier.

Ieșirile din proceduri sunt constituite din fișierele create sau situațiile de ieșire obținute și trebuie descrise, ca și intrările, cu specificatorul complet de fișier.

Logica internă de prelucrare indică modul concret de transformare a intrărilor în ieșiri (validări, conversii, actualizări, ordonări, calcule etc.). Ea se poate prezenta în proiect cu orice metodă de descriere a algoritmilor, respectiv: scheme logice, tabele de decizii, limbaj pseudocod etc.

Interfețele dintre proceduri sunt constituite din ansambluri de date (fișiere) create într-o procedură la momentul "t" care pot constitui intrări pentru aceeași procedură la momentul "t+1" sau pentru o altă procedură la momentul "t" sau "t+1".

Determinarea procedurilor se face în funcție de anumite criterii dintre care menționăm:

- ❖ tipologia și natura prelucrărilor din subsistem (actualizări, exploatare etc.);
- ❖ frecvența și termenele de realizare a prelucrărilor fișierelor de bază sau a obținerii situațiilor de ieșire;
- ❖ natura și specificul intrărilor și ieșirilor din subsistem;
- ❖ ordinea de prelucrare a fișierelor de bază;
- ❖ necesitatea asigurării unor proceduri de dirijare;
- ❖ asigurarea unor interfețe optime între proceduri etc.

Este indicat ca în proiectarea procedurilor să se folosească metoda top-down, pornindu-se de la cele două funcții principale pe care trebuie să le asigure un sistem informatic economic, respectiv actualizarea fișierelor de bază și obținerea ieșirilor proiectate.

4.5.5. Proiectarea procedurilor

Are ca obiective definirea tuturor elementelor tehnice necesare elaborării programelor, inclusiv redarea, testarea și corectarea programelor propriu-zise.

Proiectarea de detaliu a procedurilor (a programelor de aplicație) cuprinde:

- proiectarea datelor specifice procedurilor;
- proiectarea prelucrărilor specifice procedurilor.

1. Proiectarea datelor specifice procedurilor. Datele necesare procedurilor sunt concretizate în fișierele de bază ale sistemului proiectat sau în fișierele intermediare impuse de necesitatea detalierei tehnice a realizării funcției complexe a fiecărei proceduri.

Proiectarea datelor specifice procedurilor se referă la:

- a) intrările în proceduri;
- b) ieșirile din proceduri;
- c) fișierele intermediare.

a) *Proiectarea intrărilor specifice procedurilor* are în vedere datele de intrare necesare constituirii fișierelor de bază precum și dialogul procedurii cu utilizatorul, în cazul sistemelor interactive.

Datele de intrare se preiau din documentele de intrare al căror format a fost definitivat integral în proiectarea generală.

Dialogul cu utilizatorul se realizează în următoarele forme:

- prin secvențe de întrebări și răspunsuri dirijate de modulele procedurii;
- prin afișarea pe ecran a unei liste de funcții sau opțiuni de prelucrare din care utilizatorul o selectează pe cea dorită (așa numita tehnică a meniurilor).

b) *Proiectarea ieșirilor specifice procedurilor* se referă la listele (situațiile de ieșire) solicitate de beneficiar și rezultatele controlului fișierelor de bază.

Elementele necesare proiectării ieșirilor au fost definite în cadrul proiectării generale, urmând ca în proiectarea de detaliu să se stabilească modul de obținere a lor, prin programe precizate, precum și modul de dispunere în pagină sau pe monitor.

Datele care formează conținutul listelor trebuie afișate în așa fel încât să fie cât mai ușor de consultat, evitând repetarea datelor comune. Aranjarea în pagină a situațiilor de ieșire trebuie să ia în considerare restricțiile impuse de suportul utilizat, respectiv lungimea maximă a rândului și numărul de rânduri pe pagină sau pe ecran.

O soluție eficientă de obținere a situațiilor de ieșire o reprezintă utilizarea generatoarelor de rapoarte. Un generator de rapoarte este o componentă specializată a unui mediu de programare care creează rezultatele destinate afișării sau imprimării pe baza simplei lor descrieri, fără eforturi de programare.

c) *Proiectarea fișierelor intermediare* se referă la colecțiile tampon de date, diferite de cele din fișierele de bază ale sistemului, utilizate în prelucrare. Ele pot fi:

- fișiere de interfață, ce asigură comunicarea între proceduri și au structuri determinate de setul de date transmise de la o procedură la alta;
- fișiere tampon, ce asigură realizarea anumitor faze ale prelucrării, cum ar fi: sortările, indexările, asocierile etc.

2. Proiectarea prelucrărilor specifice procedurilor. Se referă la:

- a) determinarea modulelor de prelucrare;
- b) introducerea și validarea datelor;
- c) prelucrarea fișierelor;
- d) obținerea situațiilor de ieșire.

a) *Determinarea modulelor de prelucrare* urmărește descompunerea funcției complexe de prelucrare a procedurii în funcții elementare care să asigure definirea și executarea succesivă a intrărilor prelucrărilor și ieșirilor specifice procedurilor.

Modulul asigură anumite prelucrări omogene, are un nume unic și este reprezentat de o secvență finită de instrucțiuni sau comenzi de prelucrare.

Proiectarea procedurilor după metoda programării modulare constă în descompunerea procedurilor în module interconectate în conformitate cu o ierarhie bine precizată. Tehnica modularizării trebuie utilizată, atât la nivelul proiectării și realizării sistemelor informatice, cât și la nivelul fiecărei proceduri în parte. Ea presupune identificarea funcțiilor elementare ale procedurii printr-o analiză de sus în jos (top-down) și realizarea legăturilor de structura dintre module.

În cadrul unei proceduri se pot distinge mai multe tipuri de module:

- module de dirijare (module de tip monitor) care coordonează la nivelul procedurii acțiunea altor module; un modul este de tip monitor dacă în structura lui există cel puțin o operație de lansare în execuție a altui modul; un modul monitor poate să conțină numai operații de lansare în execuție a modulelor de nivel imediat inferior, sau uneori și operații care se regăsesc ca atare în algoritmul de prelucrare al procedurii respective;
- module de prelucrare (module operaționale sau module funcții) care execută operații de prelucrare conform algoritmului precizat;

- module comune folosite de mai multe module de nivel superior sau chiar de proceduri diferite;
- module nefuncționale care pot să cuprindă descrierea centralizată a datelor, comentarii etc.

b) *Introducerea și validarea datelor* asigură transpunerea datelor din documentele de intrare anterior proiectate pe un suport tehnic de date și validarea acestora.

Validarea are rolul de a controla datele de intrare pe baza unui set de reguli predeterminate, prin intermediul unor operații de control efectuate de programe sau module de validare. Operațiile de control pot avea caracter sintactic când se verifică forma, sau semantic când se verifică fondul.

Operațiile de control sintactic pot să asigure verificarea tipului datelor, a lungimii acestora, pot să controleze cheile de control pentru codurile autodetectoare și autocorectoare de erori etc.

Operațiile de control semantic pot fi:

- generale, când urmăresc depistarea unor valori incompatibile (de exemplu: luna 13, ziua 32 etc.);
- de gestiune, când urmăresc depistarea datelor care nu respectă reguli sau restricții specifice sistemului obiect (de exemplu, în cazul aplicațiilor economice: verificarea valorii și duratei de funcționare pentru ca un obiect să fie înregistrat ca mijloc fix, controlul stocurilor etc.).

Controlul semantic mai poate fi:

- local când acționează la nivelul înregistrării (de exemplu: verificarea prezenței datelor strict necesare într-o înregistrare; verificarea valorii câmpurilor ($16 \geq \text{vârsta_salariat} \leq 65$); verificarea compatibilității între valorile a două sau mai multe câmpuri (vârsta și vechimea în muncă, vârsta și numărul de copii etc.)
- global, când acționează la nivelul fișierului (de exemplu: verificarea unicității cheilor primare; verificarea numărului de înregistrări etc.).

Operațiile de validare pot fi incluse în proceduri distincte sau pot constitui module în diferite proceduri.

c) *Prelucrarea fișierelor* grupează următoarele operații:

- generarea: crearea structurii fișierului;
- inițializarea fișierului: încărcarea cu date în momentul implementării;
- actualizarea fișierului: ținerea la zi a fișierului în raport cu schimbările din sistemul obiect; pot fi actualizări simple (interactive, cu date introduse de la terminal) sau complexe (prin prelucrarea datelor înregistrate în alte fișiere)
- exploatarea (consultarea) fișierului pentru obținerea ieșirilor proiectate;
- reorganizarea fișierului prin modificarea structurii.

d) *Obținerea situațiilor de ieșire* presupune realizarea succesivă a operațiilor:

- constituirea fișierelor tampon (de interogare) cu datele necesare obținerii situației;
- ordonarea fișierului tampon;
- generarea situației (definirea structurii sale într-un program sau cu generatorul de rapoarte);
- obținerea efectivă a situației.

4.6. IMPLEMENTAREA SISTEMELOR INFORMATICE

Implementarea finalizează activitatea de realizare a sistemelor informatice. În cadrul ei se testează, se asamblează, se verifică și se asimilează de către beneficiar toate soluțiile stabilite în etapele anterioare și se validează rezultatele obținute.

Obiective ale etapei de implementare sunt: experimentarea sistemului proiectat; finisarea noului sistem; punerea în funcțiune; recepția sistemului informatic.

Principalele grupe de activități ce trebuie realizate în etapa de implementare sunt:

- asigurarea condițiilor de punere în funcțiune;
- funcționarea experimentală și punerea în funcțiune a sistemului proiectat;
- definitivarea documentației sistemului informatic;
- recepționarea și omologarea noului sistem.

4.6.1. Asigurarea condițiilor de punere în funcțiune

Implementarea sistemului informatic proiectat depinde în mod hotărâtor de felul în care beneficiarul asigură condițiile de punere în funcțiune. Aceasta presupune în principal următoarele activități: difuzarea instrucțiunilor de executare a procedurilor manuale și automate; instruirea personalului utilizator; asigurarea condițiilor organizatorice necesare; asigurarea resurselor hard; asigurarea fondului informațional.

a) Difuzarea instrucțiunilor de executare a procedurilor manuale și automate. Aceste instrucțiuni se pot grupa în instrucțiuni pentru beneficiar (personalul de specialitate - economiștii din compartimentele funcționale în sistemele economice - care va beneficia de rezultatele implementării) și instrucțiuni pentru unitatea (colectivul) de informatică ce va asigura exploatarea noului sistem.

Instrucțiunile pentru beneficiar cuprind: prezentarea succintă a aplicației și a fluxurilor informaționale și decizionale; prezentarea noilor documente proiectate și a instrucțiunilor de completare, verificare, avizare, pregătire pentru - prelucrare, și dacă este cazul, și a procedurilor de preluare pe suportii tehnici de date; prezentarea situațiilor de ieșire, a modului de folosire și interpretare a acestora .

Instrucțiunile pentru unitatea de informatică se referă la: modalitățile de prezentare și recepție a documentelor primare și a situațiilor cu rezultatele finale; modalități de pregătire și verificare a purtătorilor tehnici de date precum și de corectare a erorilor aferente; modalități de operare pe parcursul fluxului de prelucrare și de intervenție în cazul unor incidente ce pot să apară pe parcursul executării programelor.

b) Instruirea personalului utilizator grupează o serie de activități precum: inițierea în utilizarea tehnologiilor informatice (sau actualizarea cunoștințelor informatice); însușirea corectă de către fiecare persoană implicată în funcționarea noului sistem a sarcinilor ce îi revin, atât pe parcursul etapei de implementare, cât și în exploatarea curentă; pregătirea psihologică a personalului unității beneficiare pentru recepționarea corectă a efectelor pe care le va aduce cu sine sistemul informatic.

c) Asigurarea condițiilor organizatorice necesare se referă în primul rând la transpunerea în practică a modificărilor organizatorice preconizate în etapa de proiectare generală, la lansarea efectivă a documentelor proiectate și la instituirea noilor fluxuri informaționale. De asemenea, se va constitui nucleul de informaticieni ce va exploata noul sistem și se vor planifica riguros toate activitățile specifice etapei de implementare.

d) Asigurarea resurselor hard se poate efectua în trei moduri:

- prin achiziționarea tehnicii de calcul necesare, atunci când capacitatea de prelucrare poate fi acoperită cu lucrări;
- prin perfectarea accesului la o unitate prestatoare de servicii informatice pentru un anumit număr de ore-calculator, atunci când volumul lucrărilor este mic;
- printr-o soluție mixtă, care presupune rezolvarea unor probleme, de un volum mai mic, în cadrul unității, iar pentru problemele mai complexe să se apeleze la capacitatea unei unități de informatică cu o dotare corespunzătoare, lucrându-se eventual în regim de teleprelucrare.

De o deosebită importanță sunt și: asigurarea softului de bază necesar; asigurarea materialelor consumabile; asigurarea unui spațiu corespunzător desfășurării activității colectivului de informatică, care să permită realizarea unor lucrări de calitate, dar și păstrarea integrității și confidențialității fondului de date manipulat.

e) Asigurarea fondului informațional. Presupune pregătirea datelor reale și încărcarea fișierelor sau bazelor de date în vederea testării și punerii în funcțiune a noului sistem, pregătire ce se face prin: constituirea fișierelor sau bazelor de date, prin culegerea fondului informațional necesar și stocarea acestuia pe purtători tehnici de date (pentru aceasta vor fi necesare și o serie de activități pentru colectarea, ordonarea și codificarea datelor); preluarea parțială sau integrală a datelor dintr-o serie de fișiere deja existente, cu ajutorul unor programe de conversie, dacă efortul depus corelat cu timpul necesar și cu efectele obținute justifică această operație.

4.6.2. Funcționarea experimentală și punerea în funcțiune a sistemului proiectat

Dacă în unitatea beneficiară există deja un sistem informatic sau dacă implementarea noului sistem se eșalonează mult în timp pe componente funcționale, mai întâi se execută procedurile de conversie (pentru fișiere, programe, proceduri) din vechiul în noul sistem.

Responsabilitatea pentru punerea în funcțiune a unui sistem informatic revine: conducerii unității beneficiare, proiectantului sistemului informatic și colectivului de informatică care preia sarcina prelucrării datelor în noul sistem.

Punerea în funcțiune a unui sistem informatic se face în funcție de condițiile concrete din sistemul economic. Din acest punct de vedere se pot întâlni mai multe situații:

- situația punerii în funcțiune a unui sistem informatic în cazul unui sistem informațional nou, odată cu înființarea unei noi societăți comerciale sau a unei activități noi etc.
- situația punerii în funcțiune a unui sistem informatic pentru perfecționarea unui sistem informațional existent;
- situații de lansări în exploatare în funcție de modul de asigurare a tehnicii de calcul necesare (proprie; închiriată; proprie și închiriată);
- situații de punere în funcțiune diferențiate de complexitatea sistemului informatic, de sfera lui de cuprindere, de particularitățile agentului economi și de dispersarea lui în teritoriu (sunt evidente diferențierile ce există în punerea în funcțiune a unui sistem informatic la o societate comercială concentrată teritorial, față de una dispersată teritorial (unde organizarea fluxurilor informaționale ridică probleme deosebite).

Se pot folosi mai multe strategii de implementare a unui sistem informatic, în funcție de numărul și ordinea subsistemelor care se testează și de comparațiile ce se fac cu vechiul sistem.

În funcție de ordinea de testare a subsistemelor componente implementarea poate fi:

- simultană, când toate subsistemele se testează odată, fără acordarea de priorități;
- în serie, care presupune testarea pe bază de prioritate a subsistemelor, de obicei testându-se mai întâi subsistemele cu frecvența cea mai mare.

În funcție de compararea noului sistem cu vechiul sistem implementarea poate fi realizată în mai multe variante, dintre care enumerăm:

- implementarea directă cu date curente, care presupune renunțarea la vechiul sistem în vederea reducerii termenului și a minimizării cheltuielilor de implementare;
- implementarea paralelă, care se face cu date curente sau anterioare, dar o perioadă noul sistem funcționează în paralel cu cel vechi;
- implementarea pilotată, care presupune lansarea numai a anumitor subsisteme (de exemplu a celor cu frecvență maximă) folosind date din perioadele anterioare și date curente și efectuând comparații cu vechiul sistem.

De o deosebită importanță este alegerea momentului punerii în funcțiune a unui sistem informatic care trebuie să corespundă cu începutul unei perioade semnificative din activitatea organismului economic în care acesta se integrează: începutul unui an calendaristic, introducerea

unei noi forme de organizare, lansarea în producție a unei noi tehnologii, începutul unei noi activități etc.

4.6.3. Documentația finală a sistemelor informatice

În timpul implementării unui sistem informatic pot să apară modificări (structurale și funcționale) care trebuie operate și în documentația elaborată pentru a evita dificultățile în exploatarea curentă și întreținerea ulterioară. Documentația finală a unui sistem informatic se concretizează în întocmirea următoarelor lucrări: manualul de prezentare; manualul de utilizare; manualul de exploatare (operare).

Manualul de prezentare. Cuprinde concepția generală a sistemului și se adresează conducerii unității beneficiare. Un manual de prezentare conține în principal următoarele:

- elemente de identificare: pagina de titlu și semnături; cuprinsul; baza elaborării; introducerea etc;
- obiectivele, performanțele și limitele noului sistem;
- categorii de probleme abordate și domeniul de aplicare;
- schema funcțională a sistemului informatic;
- locul sistemului în ansamblul sistemului informațional al agentului economic (diagrama de relații);
- prezentarea ieșirilor: tipuri de ieșiri și modul de obținere; lista ieșirilor; exemple de ieșiri semnificative;
- prezentarea intrărilor: tipul de intrări și moduri de completare; lista intrărilor;
- structura datelor: soluția adoptată pentru organizarea și gestiunea datelor; schema de structură generală a bazei de date și a legăturilor între fișiere;
- schema generală a fluxurilor informaționale în noul sistem;
- resurse necesare exploatării sistemului informatic: configurații de calcul, resurse umane, resurse financiare, alte resurse materiale auxiliare etc;
- condiții de implementare;
- elemente de evaluare a sistemului informatic.

Manualul de utilizare. Este întocmit pentru fiecare subsistem în parte și se adresează personalului implicat în utilizarea noului sistem la unitatea beneficiară. Instrucțiunile de utilizare se elaborează pe categorii de utilizatori (compartimente, persoane), iar gruparea pe aceste categorii se face de realizatorii sistemului informatic împreună cu conducerea unității beneficiare. În principiu, un asemenea manual cuprinde:

- elemente de identificare: pagina de titlu și semnături; cuprinsul; baza elaborării; introducerea etc;
- proceduri manuale de codificare a datelor de intrare: structura codurilor pentru fiecare mulțime de date; graficul de implementare a codificării, modul de întreținere și responsabilități; liste de coduri și cifre de control; legătura cu alte sisteme de codificare, la nivel național și internațional;
- proceduri manuale de colectare și transmitere a datelor de intrare: prezentarea documentelor de intrare, a instrucțiunilor de completare, aprobare și verificare (se va anexa și câte un exemplar din fiecare document completat cu date); instrucțiuni de stocare, arhivare, termene de păstrare, difuzare, distrugere; instrucțiuni de exploatare a procedurilor destinate preluării în sistemul de calcul a datelor de pe documentele de intrare de către utilizator; proceduri de control și corecție a datelor de intrare;
- proceduri de utilizare și interpretare a ieșirilor: lista ieșirilor și prezentarea unui exemplu din fiecare situație de ieșire; instrucțiuni de interpretare și verificare; fluxuri de circulație a rapoartelor de ieșire, responsabilități, periodicitate, termene; modul de difuzare, arhivare, perioada de păstrare;

- proceduri speciale de conversie (valabile numai în etapa de implementare): lista procedurilor manuale și automate de conversie; descrierea fiecărei proceduri.

Manualul de exploatare (operare). Cuprinde informații cu privire la exploatarea efectivă a sistemului proiectat prin intermediul sistemului de calcul și se adresează personalului din unitatea de informatică. Are în componență următoarele piese:

- elemente de identificare: pagina de titlu și semnături; cuprinsul; baza elaborării; introducerea etc;
- lista procedurilor și graficul de exploatare a acestora;
- descrierea procedurilor de preluare, corectare, validare, stocare și transmitere a datelor de intrare;
- descrierea tuturor procedurilor automate;
- descrierea procedurilor de operare la calculator: operații pregătitoare pentru efectuarea lucrării; moduri de intervenție în caz de incident; lista mesajelor apărute pe parcursul operării și modul de acțiune în fiecare caz; timpul de execuție;
- proceduri de control și transmitere a ieșirilor: controlul formal și de fond al situațiilor de ieșire; modul de corectare a erorilor și de reluare a execuției; termene de predare la utilizator, evidența și responsabilitățile pentru aceste predări.

4.7. EVALUAREA SISTEMELOR INFORMATICE

Teoria și practica economică propun, în mod tradițional, aprecierea oportunității și utilității sistemelor informatice prin prisma eficienței economice.

Analiza preliminară, în faza de stabilire a necesității realizării unui nou sistem, a resurselor ce vor trebui angajate, corelată cu cea a efectelor previzibile, precum și urmărirea modului în care se cheltuiesc resursele, în concordanță cu obiectivele realizate, este fără îndoială, ca în cazul oricărei investiții, un lucru necesar dar nu suficient.

Având în vedere aceste considerente, trebuie să acceptăm ideea că evaluarea unui sistem informatic economic trebuie făcută prin prisma eficienței economice, dar mai ales prin prisma modului în care sistemul informatic își realizează obiectivele, prin prisma calității.

4.7.1. Eficiența economică a sistemelor informatice

Eficiența unui sistem informatic poate fi calculată prin compararea efectelor cu valoarea resurselor alocate pentru construirea și funcționarea sa, dar, până în prezent, nu s-a ajuns la un punct de vedere comun cu privire la metodele de cuantificare.

Determinarea resurselor consumate pentru proiectarea, realizarea și implementarea unui sistem informatic se poate face prin cumularea cheltuielilor în funcție de etapele de proiectare parcurse și categoriile de cheltuieli implicate.

În general, cheltuielile efectuate cu un sistem informatic pot fi grupate în două categorii:

a) cheltuieli de realizare, în care sunt cuprinse cheltuielile cu echipamentul, costul proiectului de realizare a sistemului informatic, costul amenajărilor auxiliare, cheltuieli cu pregătirea personalului etc.

b) cheltuieli de întreținere – exploatare, în care intră costul utilizării calculatorului dacă acesta este închiriat, salariile personalului care va exploata sistemul informatic, costul energiei electrice consumate, al hârtiei de imprimantă, discurilor flexibile, costul pieselor de schimb, costul abonamentului de întreținerea periodică etc.

Dacă din punct de vedere al elementelor de cost problemele sunt relativ clare, cuantificarea efectelor economice este mai dificilă. Efectele economice obținute prin

introducerea unui sistem informatic pot fi grupate în două categorii și anume:

a) efecte directe, din cadrul sistemului informațional cum ar fi cele rezultate din reducerea de personal, economia de formulare, rechizite etc.

b) efecte indirecte, din afara sistemului informațional, și anume cele rezultate din rezolvarea problemelor pe ansamblul unității economice.

Acest mod de abordare a efectelor economice a dus chiar la separarea eficienței economice a sistemelor informatice în eficiență economică directă și eficiență indirectă.

Teoria și practica economică recomandă o serie de indicatori folosiți în evaluarea eficienței unui sistem informatic atât în faza de proiectare cât și în faza de analiză. Acești indicatori pot fi grupați în două categorii și anume:

a) Indicatori ai efectelor economice ce se concretizează în rezultate directe și indirecte apărute în activitatea curentă a unității beneficiare (ca de exemplu: sporul de producție; sporul de profit; economia de personal; reducerea cheltuielilor etc. - toți calculați având în vedere nivelurile înregistrate înainte și după introducerea sistemului informatic);

b) Indicatorii sintetici, care cuantifică eficiența economică obținută prin introducerea sistemului informatic ca o investiție a unității economice beneficiare. Principalii indicatori în acest sens sunt:

1. Coeficientul eficienței economice, care se poate calcula pentru fiecare subsistem informatic (k_{efj}) sau pentru sistemul informatic în ansamblu, se determină cu relația:

$$k_{efj} = \frac{E_{BJ}}{C_{RJ}}$$

în care:

E_{BJ} - efectele obținute ca urmare a funcționării sistemului (subsistemului J);

C_{RJ} - cheltuieli de realizare a sistemului (subsistemului J).

2. Termenul de recuperare a cheltuielilor totale cu realizarea unui subsistem informatic (T_{ij}) sau a sistemului informatic în ansamblu, care se exprimă în ani și se determină astfel:

$$T_{ij} = \frac{1}{k_{efj}} = \frac{C_{RJ}}{E_{BJ}} \text{ (ani)}$$

3. Coeficientul eficienței comparate (k_{ecomp}), calculat astfel:

$$k_{ecomp} = \frac{E_{B2} - E_{B1}}{C_{R2} - C_{R1}}$$

în care:

E_{B2} , E_{B1} - efectele obținute în varianta de sistem informatic care presupune investiție suplimentară, respectiv în varianta cu care se compară;

C_{R2} , C_{R1} - cheltuielile cu realizarea variantei de sistem informatic care necesită investiții suplimentare, respectiv cheltuielile cu realizarea variantei cu care se compară.

4. Termenul de recuperare a investiției suplimentare (T_{rs}), exprimat în ani, care se determină cu relația:

$$T_{rs} = \frac{1}{k_{ecomp}} = \frac{C_{R2} - C_{R1}}{E_{B2} - E_{B1}} \text{ (ani)}$$

4.7.2. Managementul calității sistemelor informatice

În aprecierea calității unui sistem informatic trebuie pornit de la realitatea evidentă că sistemul este construit pentru client. Premisa de bază a calității sistemului informatic este aceea că ea trebuie abordată sistemic, începând cu fundamentarea deciziei de realizare a unui nou sistem, continuând cu organizarea și conducerea realizării sistemului, cu implementarea acestuia, pentru ca în final să putem aprecia calitativ funcționarea efectivă a sistemului informatic.

Calitatea unui sistem informatic se poate privi din două puncte de vedere:

- pe de o parte, din punctul de vedere al utilizatorilor sistemului (aspectul funcțional);

- pe de altă parte, din punctul de vedere al celor ce au realizat sistemul (aspectul tehnic).

Din punct de vedere funcțional, ceea ce pare tehnic rațional și corect poate să fie nefolositor utilizatorului.

Aspectul tehnic pornește de la premisa că, date fiind condițiile dintr-o economie concurențială, realizatorul unui sistem informatic trebuie să aleagă cele mai noi, cele mai performante soluții.

Absolutizarea unuia sau altuia dintre cele două aspecte poate să creeze serioase greutăți în exploatarea sistemului informatic. Trebuie avut în vedere că sistemele informatice economice sunt create de oamenii pentru a fi utilizate în organizații de către oameni.

În literatura de specialitate se remarcă existența a două concepții cu privire la calitate:

- *una orientată tehnic către producție*, care cercetează existența unor abateri ale caracteristicilor produsului realizat față de nivelurile prevăzute în documentația tehnică întocmită înainte de începerea realizării produsului;

- *una orientată funcțional către utilizator*, care consideră calitatea ca fiind *aptitudinea unui produs sau a unui serviciu de a satisface nevoile utilizatorului*.

Problema calității materialelor (echipamentelor) utilizate este o problemă clasică de management al calității industriale, în timp ce problema elementelor logice este specifică domeniului. Norma ISO 9126 a enunțat șase caracteristici care permit descrierea calității unui sistem informatic:

- *Capacitatea funcțională* care cercetează dacă funcțiile pe care sistemul le realizează corespund nevoilor explicite sau implicite. Ea este descompusă în cerințele: operabilitate, conformitate cu regulamentele, respect față de norme etc.

- *Facilitatea în utilizare* este definită în raport cu ansamblul utilizatorilor potențiali și se referă la efortul depus pentru a utiliza sistemul. Poate fi descompusă în facilități de comprehensiune, de inițiere, de exploatare etc.

- *Fiabilitatea* este capacitatea sistemului de a-și menține, în condițiile specifice utilizatorului, parametrii de performanțe pe toată perioada de exploatare stabilită și se descompune în: maturitate, toleranță la erori, posibilități de recuperare etc.

- *Randamentul (eficiența)* cercetează raportul între serviciile oferite de sistem (efectele sistemului) și resursele utilizate într-un cadru dat.

- *Mentenabilitatea* care se referă la eforturile necesare pentru a menține și dezvolta sistemul, respectiv la posibilitățile de a aduce corecții sau ameliorări funcționale ulterioare, de a efectua schimbări organizaționale sau de documentație etc. Este descompusă în: facilități de analiză, modularitate, stabilitate, facilități de testare etc.

- *Portabilitatea* definită ca fiind capacitatea sistemului informatic de a fi transferat pe un mediu tehnologic diferit din punct de vedere fizic sau logic. Această caracteristică este descompusă în facilități de instalare și facilități de adaptare.

Pentru perfecționarea sistemelor informatice, pe lângă aprecierile făcute de specialiști în urma unei analize temeinice este nevoie și de un sistem de indicatori care să permită estimarea calității. Pentru aceasta, se pot avea în vedere următorii indicatori:

- numărul de erori detectate în faza de implementare;
- numărul de erori detectate în exploatare;
- media gravității erorilor detectate
- intervalul mediu de timp necesar pentru remedierea unei erori:
- timpul mediu de acces la informații
- continuitatea în funcționare
- numărul informațiilor eronate rezultate din sistem,
- numărul cererilor clienților pentru care sistemul nu a fost capabil să dea un răspuns,
- timpul mediu necesar utilizatorilor pentru a învăța să lucreze cu sistemul,
- numărul comenzilor pe care utilizatorul trebuie să le rețină etc.

CAP.5. PROGRAMARE ÎN INTERNET

5.1. INTERNET ȘI WEB. ARHITECTURI DE SISTEME DISTRIBUITE

5.1.1. Noțiuni generale

Cea mai răspândită rețea publică este Internet-ul. Internet-ul poate fi descris ca un *sistem deschis*, adică un sistem a cărei arhitectură nu este secretă (producătorii săi i-au făcut publică structura suficient de detaliat astfel încât alți dezvoltatori să-și poată interfata la el propriile produse). Câteva exemple de sisteme deschise sunt: sistemul de operare UNIX și utilitățile asociate, protocoalele rețelei Internet. Putem considera ca sistemele deschise au inspirat de asemenea tehnologiile bazate pe Java și XML. Exemple de sisteme proprietare sunt: sistemele Microsoft Windows și Mac OS cu utilitățile aferente.

Internet-ul are o arhitectură multinivel, inspirată de modelul de referință ISO/OSI.

În 1990 Tim Berners Lee a creat World Wide Web (WWW, WEB sau W3). Acesta a propus:

- O metodă de a atasa nume simbolice calculatoarelor din Internet
- O metodă de reprezentare a documentelor cu legături simbolice între ele (HTML)
- Conceptele de *server WEB* pentru stocarea acestor documente și *navigator* (engl.*browser*) pentru afișarea acestor documente.

Internet-ul este o rețea de sub-rețele. Sub-rețelele comunică între ele printr-un nod special numit *gateway*.

Protocoale ale rețelei Internet:

- Telnet – protocol ce permite accesul la distanță la un calculator conectat la Internet, cu condiția ca utilizatorul să aibă drept de acces la calculatorul respectiv
- FTP și TFTP – protocoale de transfer de fișiere între calculatoare
- SMTP – protocol pentru transferul poștei electronice între calculatoare
- Kerberos – protocol pentru transferul confidențial de date între calculatoare
- SNMP – protocol pentru monitorizarea și administrarea rețelelor de calculatoare
- DNS – protocolul care permite denumirea simbolică a calculatoarelor conectate la Internet
- NFS – colecție de protocoale care permite accesul transparent la fișierele și directoarele dintr-o rețea de calculatoare
- TCP – protocol orientat pe conexiune ce permite transferul sigur și fiabil al datelor între procesele aplicațiilor ce rulează pe calculatoare conectate la Internet
- UDP – protocol neorientat pe conexiune cu funcție oarecum similară cu TCP, doar că nu garantează transferul sigur al datelor
- IP – protocol ce asigură transferul pachetelor între calculatoarele conectate la Internet
- ICMP – protocolul de transfer a informațiilor de comandă și eroare între componentele rețelei
- ARP și RARP – protocoale ce asigură corespondența directă și inversă între adresele hardware și adresele Internet ale calculatoarelor conectate la Internet

Adresarea în Internet

Principala funcție a Internet-ului este schimbul de date între calculatoare. Pentru aceasta, fiecare calculator din Internet are asignată o *adresa IP*.

Versiunea cea mai răspândită la ora actuală a protocolului IP este IPv4. În IPv4 adresa unui calculator este un șir de 32 de biți. Acest șir se reprezintă sub forma unui 4-uplu format din 4 octeți, separați prin câte un punct. Exemplu: 151.23.40.3. Există și versiunea IPv6 care folosește 128 de biți. Multimea de adrese IP este împărțită în 5 clase:

- Clasa A, 0 adresa rețea (7) adresa calculator (24)
- Clasa B, 10 adresa rețea (14) adresa calculator (16)
- Clasa C, 110 adresa rețea (21) adresa calculator (8)
- Clasa D, 1110 adresa multicast (28)
- Clasa E

Adresa 127.0.0.1 se mai numește și adresă de *loopback*. Datele trimise aici se întorc înapoi la sursă. Aceasta este utilă pentru testarea aplicațiilor de rețea folosind un singur calculator.

Adresele IP sunt dificil de memorat în format numeric, de aceea s-a introdus o metodă de a le atașa nume simbolice prin sistemul de numire a domeniilor (engl. *domain name system* – DNS). DNS reprezintă o modalitate de a denumi simbolic calculatoarele dintr-o rețea bazată pe TCP/IP (Internet) folosind o schemă de denumire ierarhică. Fiecare sufix dintr-un nume se numește *domeniu*. Determinarea adresei IP pornind de la nume se face prin interogarea unui *server de nume* (engl. *name server*). DNS reprezintă multimea tuturor acestor servere.

Clienți și servere

Un *server* este un calculator (program în execuție) din cadrul unei rețele care furnizează servicii altor calculatoare (programe în execuție) din cadrul rețelei. Un *client* este un calculator (program în execuție) dintr-o rețea care beneficiază de serviciile unui calculator (sau program) server.

În Internet, comunicarea între un client (program) și un server (program) se face folosind suita de protocoale TCP/IP. Aceasta se bazează pe noțiunile de *port* și *soclu* (engl. *socket*). Un *port* reprezintă un canal abstract prin care un calculator (program care rulează pe calculator) poate comunica cu exteriorul. Un port este identificat printr-un număr. Porturile care aparțin intervalului 0...1023 sunt rezervate pentru servicii speciale. Câteva dintre acestea sunt:

- Port 7: ECHO
- Port 21: FTP
- Port 23: Telnet
- Port 80: HTTP
- Port 25: SMTP
- Port 110 :POP3
- Port 150 :SQL-NET

Un *soclu* este o pereche formată dintr-o adresă de IP și un port. Un soclu abstractizează noțiunea de canal de comunicație într-o rețea bazată pe TCP/IP, usurând astfel programarea.

Tipuri de servere

- Servere de fișiere. Furnizează fișiere la cererea clientului; spre exemplu un depozit de documente (engl. *document repository*).
- Servere de baze de date. Stocază colecții mari de date structurate sub forma unor baze de date; furnizează servicii de interogare a acestora folosind SQL.
- Servere de *groupware*. *Groupware* = un sistem care permite unui grup de participanți să lucreze împreună într-un mediu partajat.

- Servere WWW. Sunt servere de fișiere care conțin componentele unui site din WWW. Accesul la ele se face printr-un program client special numit *navigator*.
- Servere de posta electronică. Permit recepția, stocarea și trimiterea de mesaje prin posta electronică.
- Servere de obiecte. Stocază obiecte și permit programelor client să trimită mesaje acestor obiecte.
- Servere de imprimare. Furnizează clienților servicii de imprimare.
- Servere de aplicații. Sunt servere dedicate uneia sau mai multor aplicații particulare și conțin programele dedicate aplicației respective.

5.1.2. World Wide Web

WWW este un *sistem hipermedia distribuit*. Se bazează pe un model de structurare a documentelor care folosește trei concepte:

- **Multimedia** – se referă la integrarea mai multor tipuri de media în cadrul aceluiași model de document: text, grafică, imagine, video, etc.
- **Hiperdocument** – se referă la crearea de legături între documente, folosind un mecanism propriu modelului de document.
- **Documente distribuite** – se referă la documente care conțin legături la documente stocate pe alte calculatoare din cadrul unei rețele.

Se spune că WWW folosește un *model de documente hipermedia distribuite*. Termenul de hipermedia înglobează conceptele de multimedia și hiperdocument.

Fiecare autor care creează o resursă informațională (engl. *information resource*) în WWW o consideră ca fiind un document separat. Însă putem considera că, la nivel global, multimea tuturor resurselor informaționale din WWW formează un unic document hipermedia distribuit. Din acest punct de vedere, termenul de resursă informațională este mai potrivit decât cel de document. Există trei nivele de distribuție a resurselor informaționale în WWW: același fișier, fișiere separate pe același calculator, calculatoare diferite.

Deși WWW a apărut în 1990, conceptele de bază au rămas aceleași până astăzi: URL (denumirea unui document), HTTP (regăsirea unui document) și HTML (descrierea conținutului unui document).

Arhitectura WWW este o arhitectură client/server tipică și are:

- Un server WWW are sarcina de a gestiona o mulțime de documente din cadrul WWW. Aceste documente se numesc și *pagini WWW*.
- Un client generic de WWW este un program care emite cereri către un server WWW pentru accesarea paginilor WWW gestionate de acel server. Exemple de clienți sunt:
 - *Un navigator WWW care permite regăsirea și afișarea paginilor WWW în scopul vizualizării conținutului lor de către un agent uman.*
 - *Un program de tip softbot care localizează diverse pagini WWW în scopul creării unui index. Indexul poate fi utilizat ulterior de un motor de căutare.*

Conceptele pe care se bazează tehnologia WWW sunt:

- Schema de denumire a resurselor (engl. *uniform resource locator*) URL
- Protocolul de transfer al documentelor (engl. *hypertext transfer protocol*) HTTP
- Limbajul de specificare a conținutului paginilor WWW (engl. *hypertext markup language*) HTML

Pentru identificarea resurselor în WWW se folosește un URL (engl. *Uniform Resource Locator*). Un URL este un identificator simbolic al resursei și este compus din două părți:

- Schema, care indică modalitatea folosită pentru denumirea resurselor. Spre exemplu, schema poate fi numele unui protocol: *ftp*, *http*, etc
- Partea specifică schemei, care indică cum se adresează resursa în cadrul schemei respective

Sintaxa URL este **schema “:” parte-specifica-schemei**. Partea specifica schemei are sintaxa “//” [utilizator [“:” parola] “@”] gazda [“:” port] “/” cale

- *utilizator* și *parola* sunt optionale și se aplica doar cu schemele care au sens (de exemplu *ftp*).
- *gazda* indica numele calculatorului pe care se afla resursa, sau adresa de IP a acestuia.
- *port* reprezinta numarul portului pe care se face conexiunea. Este optional, deoarece acest numar este predefinit pentru serviciile standard. Pentru HTTP portul predefinit este 80.
- *cale* reprezinta calea de acces la resursa în cadrul calculatorului specificat. În general este o cale fizica existenta în sistemul de fisiere de pe calculatorul gazda.

Exemple de aplicatii WWW:

- Furnizarea de adrese de email anonime pentru trimiterea de *spam*. Termenul de *spam* se refera la trimiterea de mesaje de email cu anunțuri unor receptori ce nu au solicitat primirea mesajelor respective. *Spam*-ul este detestat de toti utilizatorii de Internet.
- Site-uri de verificare automata a validitatii/actualitatii unor legaturi WWW.
- Site-uri de arhivare pentru stocarea sigura de fisiere.
- Site-uri care furnizeaza adrese de email gratuite. Sunt utile pentru utilizatorii mobili.
- Motoare de cautare (eng.*search engines*). Cautarea se face într-o baza de date uriasa care conține informații despre documentele din WWW, organizate sub forma unui index. Specificarea cautarii se face printr-o interogare (engl.*query*). Baza de date este construita cu ajutorul unui program special numit *spider* (un client special de HTTP) care navigheaza automat pe WWW și stocheaza în indexul local informațiile relevante gasite. *Spamdexing* este o tehnica folosita de unele site-uri pentru a fi gasite ca relevante la cautarea unor cuvinte cheie raspandite.
- Grupuri de utilizatori interesati de un subiect comun – *newsgroup* și *mailing list*.
- Aplicatii de comert electronic

5.1.3. Protocolul HTTP

HTTP este protocolul de comunicare între clientii și serverele WWW. El specifica cererile care pot fi adresate de clienti și raspunsurile care pot fi generate de servere. Specificatia conține *structura* și *formatul* (sintaxa) acestora.

Cererile și raspunsurile HTTP se mai numesc și *mesaje*. Un mesaj este un sir de caractere ce conține: *linia de start* (engl.*start line*), zero sau mai multe *campuri antet* (engl.*message header field*) și optional un *camp corp* (engl.*message body field*).

Antetele unui mesaj pot fi: *antete generale* (engl.*general header*) – se refera la mesaj, nu la entitatea transmisa; *antete de entitate* (engl.*entity header*) – se refera la entitatea transmisa sau referita; *antete raspuns* (engl.*response header*) – serverul comunica clientului informații care nu au fost incluse in linia de start.

Cereri – Linia de start se numeste *linie de cerere* (engl.*request line*) și are structura:
metoda spatiu url_resursa spatiu versiune_http sfarsit_linie

Raspunsuri – Linia de start se numeste *linie de stare* (engl.*status line*).

Referitor la sintaxa formală a unei cereri HTTP, trebuie precizate urmatoarele:

- HTTP 1.0 este descris la nivel informațiv în documentul RFC 1945.
- HTTP 1.1 este descris în propunerea de standard RFC 2068.
- O *cerere* HTTP are structura urmatoare ([...] specifica un element optional și * specifica 0 sau mai multe repetitii ale unui element):

cerere =

linie-de-cerere
(*antet-general* | *antet-cerere* | *antet-entitate*)*
sfarsit-de-linie
[*corp-mesaj*]
linie-de-cerere = *metoda spatiu uri spatiu versiune-http sfarsit-de-linie*
metoda = OPTIONS | GET | HEAD | POST | PUT | DELETE |
TRACE | CONNECT | *metoda-extensie*
uri = * | *uri-absolut* | *cale-absoluta*

– Un raspuns HTTP are structura urmatoare:

raspuns =

linie-de-stare

(*antet-general* | *antet-raspuns* | *antet-entitate*)*

sfarsit-de-linie

[*corp-mesaj*]

linie-de-stare = *versiune-http spatiu cod-stare spatiu fraza-motiv sfarsit-de-linie*
cod-stare = *cod-succes* | *redirectare* |

eroare-client | *eroare-server* | ...

cod-succes = *ok* | *creat* | *acceptat* | ...

ok = 200

creat = 201

acceptat = 202

redirectare = *mutat-permanent* | *mutat-temporar* | ...

mutat-permanent = 301

mutat-temporar = 302

eroare-client = *cerere-gresita* | *neautorizat* | *plata-necesara* | *interzis* |

resursa-inexistenta | *metoda-nepermisa* | ...

cerere-gresita = 400

neautorizat = 401

plata-necesara = 402

interzis = 403

resursa-inexistenta = 404

metoda-nepermisa = 405

eroare-server = *eroare-interna* | *nu-este-implementata* | ...

eroare-interna = 500

nu-este-implementata = 501

Cookies

HTTP este un protocol *fara stare* (engl.*stateless*). Acest lucru înseamna ca HTTP nu dispune de un mecanism propriu care sa permita unui server WWW sa poata pastra informații despre starea utilizatorului. Termenul de *cookie* se refera la mecanismul prin care o aplicatie WWW pe partea de server poate stoca și regasi informații pe partea de client. El permite adaugarea unei stari a conexiunii stocata la client. Serverul specifica clientului ca vrea sa stocheze un *cookie* prin antetul Set-Cookie, în maniera urmatoare:

Set-Cookie: *Nume=Valoare*

Ulterior, clientul va include cookie-ul într-un antet al unei cereri sub forma:

Cookie: *Nume=Valoare*

Suplimentar, antetul Set-Cookie mai poate conține urmatoarele informații:

- domain: specifica domeniul în care se aplica cookie-ul
- expires: specifica data de expirare a cookie-ului
- path: specifica URL-urile la care clientul va returna cookie-ul în cererea HTTP
- secure: specifica faptul ca cookie-ul va fi returnat de client numai dacă conexiunea este sigura.

Ex: Set-Cookie: Credit=111; secure; expires=Thursday, 07-Dec-2000, 10:00:00 GMT; domains=.comp-craiova.ro; path=/

Relativ la trimiterea de date către un server WWW, trebuie precizat ca :

- Paginile de WWW pot fi *statice* și *dinamice*. *Paginile statice* sunt stocate explicit pe server și returnate clientilor ca raspuns la cereri GET. *Paginile dinamice* sunt generate dinamic de server, nefiind stocate explicit. Este posibil ca aceste pagini sa fie configurate pe baza unor date trimise de la client către server. Astfel de date pot fi de exemplu criterii de cautare într-o baza de date, paginile generate dinamic fiind în acest caz rapoartele rezultate în urma cautarii.
- Exista doua metode de a transmite date de la client la server:
 - Folosind metoda GET, datele sunt atasate URL-ului sub forma unor perechi variabila-valoare. În acest caz URL-ul reprezinta un program aflat pe server și datele sunt transmise acestui program în mediul sau de executie (engl.*environment*). La URL se adauga un sir de forma:
 $?nume_1=valoare_1\&nume_2=valoare_2\& \dots \&nume_n=valoare_n$
Un ' ' este codificat prin '+' și un caracter special printr-un cod hexa precedat de '%'. Ex. $?cale=\%2Fweb\%2$ semnifica valoarea /web/ pentru variabila *cale*.
 - Folosind metoda POST, datele sunt atasate în corpul cererii, spre deosebire de cazul anterior, unde sunt atasate URL-ului, în antet. Ele sunt transmise programului prin intrarea standard.
- Serverul preia datele de la programul invocat prin iesirea standard și le trimite apoi clientului. Aceasta tehnica de a extinde funcționalitatea unui server WWW pentru generarea dinamica de pagini se numeste Common Gateway Interface (CGI).

Middleware

Middleware este un termen destul de vag ce se referă în general la toate nivelurile software intermediare care sprijină comunicarea dintre un client și un server.

Un middleware furnizează un set standard de interfețe pentru o colecție de resurse distribuite disparate, eterogene și proprietare. Astfel dezvoltatorii își vor interfața aplicațiile cu partea de middleware în loc de interfețele de nivel coborât ale resurselor proprietare. Un exemplu de middleware este software-ul care interfațează un program navigator de sistemul WWW.

O tehnologie foarte răspândită este middleware-ul orientat pe mesaje – MOM (engl. *message-oriented middleware*). MOM gestionează tranzacțiile dintre un client și un server prin intermediul unor cozi care stochează mesajele transmise între clienți și serveri.

Un exemplu de MOM este *WebSphere MQ* dezvoltat de IBM (fost *MQSeries*). *WebSphere MQ* gestionează transferul de mesaje între clienți și serveri și știe să prelucreză patru tipuri de mesaje: datagrame – mesaje unidirectionale, mesaje de cerere, mesaje de răspuns și mesaje de raportare. *WebSphere MQ* este un lider în domeniul platformelor middleware pentru integrarea aplicațiilor de e-business.

5.1.4. Arhitecturi multistrat (engl. *tiered architectures*)

O *aplicație distribuită* este compusă dintr-o multitudine de programe care rulează pe mai multe calculatoare conectate în rețea. O schiță a acestor programe împreună cu calculatoarele (engl. *hosts*) pe care rulează, responsabilitățile lor și protocoalele prin care comunică se numește *arhitectura distribuită*.

O clasificare a arhitecturilor distribuite se bazează pe conceptul de *strat* (engl. *tier*). Un strat poate fi un calculator (înșă putem avea aplicații distribuite virtual care rulează pe un același calculator) sau o partiție logică de prelucrare din cadrul aplicației. De obicei un strat corespunde unui client sau server.

Avantaj: paradigmele de codificare sunt diferite de la strat la strat astfel ca straturi diferite cer îndemânări de programare diferite

Partiționarea logică a unei aplicații distribuite trebuie să aibă în vedere cel puțin următoarele trei elemente:

- Logica de prezentare
- Logica problemei (engl. *business logic*)
- Logica datelor, responsabilă cu persistența datelor, controlul accesului concurent, corectitudinea tranzacțiilor, etc

Avantaj: straturile permit separarea elementelor enumerate mai sus

Clasificarea arhitecturilor multistrat:

- 1 strat (engl. *one tier*)
 - Este simplă deoarece nu există conectare în rețea
 - Performanțe bune, deoarece nu există comunicații în rețea
 - Sistemul este autoconținut
 - Nu există posibilitatea accesului de servicii la distanță
 - Arhitectura monolitică, deci potențial de “cod spaghetti”
- 2 straturi (engl. *two tiers*)
 - Straturi: client și server de WWW, arhitectura oarecum simplă
 - Separa logica de prezentare de logica problemei
 - Potențial mic pentru partajarea resurselor, o problemă în comerțul electronic
- 3 straturi (engl. *three tiers*)
 - Straturi: client, server de WWW, server de baze de date
 - Separa logica de prezentare, logica problemei și logica datelor
 - Necesită expertiză în plus, mapearea obiectual-relațională este destul de dificilă

- 4 straturi (engl.*four tiers*)
 - Straturi: client, server WWW, server de aplicatii, server de baze de date
 - Flexibilitate ridicata, practic poate realiza orice
 - Nivel de expertiza foarte înalt, curba de învățare mare, cost foarte ridicat, poate fi ineficienta datorita generalitatii
 - Partea dintre logica de prezentare (client) și logica datelor (baza de date) se mai numeste și strat intermediar (engl.*middle layer*). El conține printre altele obiectele problemei (engl.*business objects*), ce corespund entitatilor din domeniul problemei

Putem avea și arhitecturi cu $n \geq 4$ straturi (engl.*n-tiers*). Cu cat numarul de straturi este mai mare, cu atat performantele pot sa scada, implementarea este mai dificila și cere expertiza mai mare, complexitatea sistemului este mai mare, costul total al sistemului creste. În consecinta, stabilirea numarului de straturi trebuie facuta cu grija, în funcție de cerintele reale ale aplicatiei. Cel mai adesea o arhitectura în 3 straturi este suficienta. Serverele de aplicatii sunt în general foarte scumpe și curba de învățare este foarte lenta.

5.1.5. Paradigme de programare distribuita

1. Modelul transferului de mesaje:

- Se bazeaza pe ideea de *protocol*. Acesta poate fi gandit ca un limbaj ce întruchipeaza funcțiile cerute de un client și pe care le furnizeaza un server.
- Protocoalele se impart în:
 - *Protocoale fixe*, al caror vocabular este fixat și incapsulat în codul serverului și clientului.
 - *Protocoale adaptive*, care se pot schimba la momentul executiei
- Un exemplu de protocol fix este cel pentru comunicarea cu un server de nume:
 - Partea de client: CAUTA Nume, STERGE Nume, ADAUGA Nume, Resursa, MODIFICA Nume, Resursa
 - Partea de server: RESURSA Detalii, STERGE OK, ADAUGA OK, MODIFICA OK, EROARE Cod
- O metoda de a implementa protocoalele adaptive este folosirea obiectelor serializabile. Acestea sunt obiecte care pot fi transferate în retea sub forma de date. Un astfel de obiect poate conține funcționalitatea unei noi comenzi din cadrul protocolului.
- În cadrul acestui model clientii și serverii comunica prin transfer de mesaje de-a lungul unor canale de comunicatie. Corespunde oarecum cu structura rețelei în care ruleaza clientii și serverii. Scopul acestui model este de a abstractiza detaliile de nivel coborat și de a face astfel programarea aplicatiilor mai usoara.
- Se preteaza în urmatoarele situatii:
 - Cerintele de comunicare sunt foarte simple
 - Sunt necesare performante foarte bune; acest model este cel mai eficient dintre modele discutate; plata pentru aceasta eficienta este cresterea complexitatii programarii
- Exemplu: HTTP, protocolul de comunicare cu un server de WWW
- Exista doua clase:
 - Transfer *sincron* de mesaje: entitatea A trimite un mesaj către entitatea B, în timp ce entitatea B prelucreaza mesajul, entitatea A se opreste și asteapta un raspuns, după ce entitatea B raspunde entitatea A isi conținea activitatea
 - Transfer *asincron* de mesaje: entitatea A trimite un mesaj către entitatea B, in timp ce entitatea B prelucreaza mesajul, entitatea A isi conținea activitatea, cand entitatea B raspunde, entitatea A poate prelua mesajul.

2. Modelul obiectelor distribuite:

- Se numeste *obiect distribuit* un obiect rezident pe un calculator care poate fi invocat de obiecte rezidente pe alte calculatoare astfel încat, dpdv al programatorului, obiectele sa para a

fi situate pe un același calculator (adică toate detaliile de comunicare să fie ascunse programatorului).

- Tehnologia obiectelor distribuite presupune următoarele:
 - Interceptarea apelurilor către obiecte
 - Localizarea obiectelor
 - Comunicarea mesajelor și parametrilor către aceste obiecte
 - Optional, returnarea eventualelor rezultate și transmiterea lor obiectului apelant
- Avantaj: obiectele distribuite corespund 100% modelului orientat pe obiect și din acest motiv trecerea de la proiectare la implementare este ușoară
- Dezavantaj: performanțele tehnologiilor de obiecte distribuite sunt inferioare tehnologiilor de transfer de mesaje
- Exemple
 - CORBA, propus de OMG
 - Apelul metodelor la distanță (engl. *Remote Method Invocation*) Java RMI
 - DCOM, propus de Microsoft

3. Modelul evenimentelor:

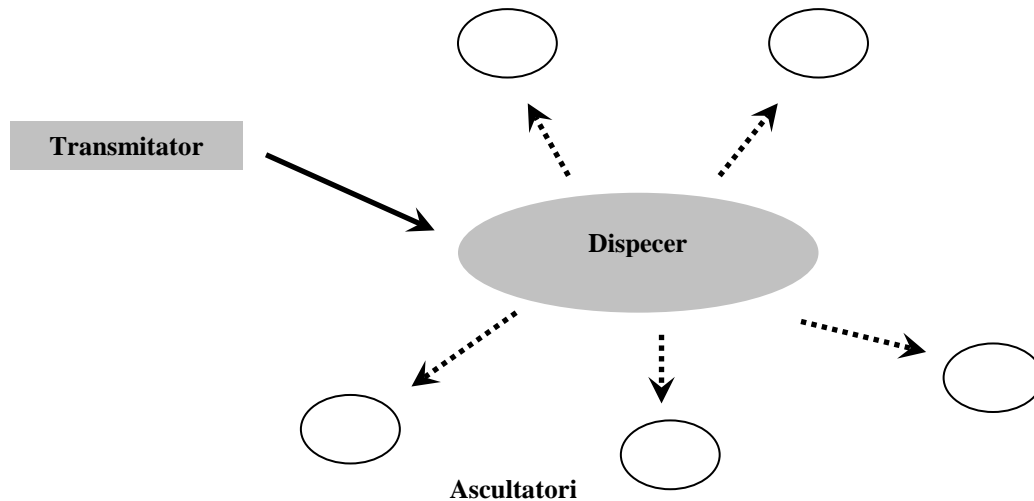
- Acest model presupune asocierea unor porțiuni de cod cu anumite evenimente. Codul va fi executat automat la declansarea evenimentelor respective.
- Acest model de programare este foarte răspândit în programarea interfețelor grafice. Dezvoltarea unei astfel de interfețe presupune următoarele:
 - Controale grafice, spre exemplu butoane, sunt plasate într-un container de tip fereastră principală, numită și cadru (engl. *frame*)
 - Fereastră cadru implementează o interfață prin intermediul căreia va fi invocată la declansarea unor evenimente. Codul de tratare a evenimentelor va fi amplasat în metodele ce implementează această interfață.
 - Fereastră cadru se înregistrează la controalele grafice ca obiect ascultător (engl. *listener*), ceea ce semnifică faptul că este interesată de a fi informată la declansarea anumitor evenimente.
 - La declansarea unui eveniment, controlul grafic informează toate obiectele ascultător înregistrate la el.

Magistrala de obiecte:

- Modelul evenimentelor este folosit și în arhitecturile de tip *magistrala de obiecte* (engl. *object bus*)
- Într-o magistrală de obiecte, serverele împing datele către clienți, din acest motiv folosindu-se și terminologia *push tehnology*. Spre deosebire de aceasta, în obiectele distribuite clienții extrag datele de la serveri, folosindu-se terminologia *pull technology*.
- În magistrala de obiecte, transmitatorul este server și ascultătorii sunt clienți.
- Arhitectura magistrală de obiecte este utilă în aplicațiile în care evenimentele se declanșează în timp real iar mulțimea de ascultători se poate schimba dinamic. Exemple:
 - Furnizarea de date despre piața de capital instituțiilor financiare interesate
 - Teleconferințe sau aplicații conversaționale (engl. *chat room*), unde mesajele între participanți se transmit în timp real
 - Aplicații multimedia distribuite de tip video la cerere (engl. *video on demand*), unde un mare volum de date trebuie transmis în timp real abonatilor
- Există două mari clase de arhitecturi de tip magistrală de obiecte:
 - Arhitectura *butuc și spate* (engl. *hub and spoke*)
 - *Magistrala cu multitransmisie* (engl. *multicast bus architecture*)

Arhitectura butuc și spite:

- Transmisorul trimite date către dispecer (butuc), iar acesta le distribuie ascultătorilor înregistrați. Poate exista câte un dispecer pe canal sau un singur dispecer pentru toate canalele.
- Avantaje: ușor de implementat folosind socket-ii sau RMI, contabilizarea poate fi centralizată la dispecer
- Dezavantaje: se generează un volum mare de trafic comparativ cu magistralele cu multitransmisie, fiabilitate scăzută deoarece totul depinde de dispecer



Magistrala cu multitransmisie:

- Tehnica multitransmisiei permite transferul unui singur mesaj de la transmitator către mai mulți receptori.
- Mesajul este transmis pe o magistrală, de unde este preluat de toți ascultătorii interesați; ascultătorii sunt activați printr-un eveniment care îi informează că mesajul este disponibil pe magistrală.
- Un exemplu este *iBus* de la *SoftWired Ltd* care poate fi gândită ca un fel de implementare software a protocolului Ethernet. Astfel mesajele vor fi preluate numai de destinatarii interesați; dacă mesajul nu este necesar este pur și simplu ignorat și lăsat să treacă următorului destinatar conectat la magistrală.

4. Modelul tuplelor:

- A fost folosit în limbajul de programare de nivel înalt Linda, dezvoltat de cercetătorii americani Nicolas Carriero and David Gelerntner în 1980. Acesta a fost răspândit doar în mediul academic. Firma Sun s-a inspirat din el în 1990 pentru a dezvolta tehnologia JavaSpaces ca parte a proiectului JINI. JINI este o arhitectură deschisă pentru interconectarea în rețea a unei multimi de servicii implementate soft sau hard și care poate fi reconfigurabilă dinamic, iar JavaSpaces este modelul de programare distribuită folosit în JINI.
- În JavaSpaces procesele comunica prin intermediul unor zone partajate numite *spatii* (engl. *spaces*). Clienții pot accesa aceste spații prin 3 operații:
 - *write*, pentru adăugarea unui nou obiect la un spațiu
 - *take*, pentru citirea unui obiect și eliminarea sa dintr-un spațiu
 - *read*, pentru crearea unei copii a unui obiect dintr-un spațiu
- Spațiile sunt containere de obiecte cu următoarele proprietăți:
 - Sunt partajate și pot fi accesate concurent
 - Sunt persistente

- Sunt asociative adica obiectele sunt localizate prin *regasire asociativa* Tranzactiile pe aceste spatii sunt sigure
- Spatiile permit schimbul de conținut executabil

5.2. PROGRAMARE PE PARTE DE CLIENT

Termenul de *programare pe partea de client* (engl.*client side programming*) se refera la executarea de programe la client în scopul cresterii gradului de interactivitate al paginilor WWW. Programele destinate a fi executate la client se pot transmite de la server către client fie în *format sursa*, fie în *format obiect*.

Termenul de *programare pe partea de server* (engl.*server-side programming*) se refera la executarea de programe la server în scopul de a genera date și/sau rapoarte care sunt trimise clientului sau mai general, pentru extinderea funcționalitatii unui server de WWW. Datele și rapoartele sunt trimise clientului de obicei în format HTML.

5.2.1. Limbajul HTML

Limbajul HTML reprezintă *Lingua franca* pentru publicarea de hipertext în pagini WWW. HTML este inspirat din SGML - un *limbaj de meta-marcare* (engl.*meta-markup*). În esenta, acesta permite definirea de noi *tipuri de documente* printr-o *definiție de tip de document* - DTD. Unui tip ii corespunde un *limbaj de marcarea* specific (ex. HTML), ce descrie conținutul și structura unui document. Un document scris într-un limbaj de marcarea conține:

- *Date* = conținutul propriu-zis al documentului.
- *Marcaje* = informații referitoare la structura documentului. Se numesc și *metadate*.

Versiunea 4.0 definește 3 tipuri de documente HTML:

- Documente *tranzitionale*: acest tip se va folosi doar pentru verificarea documentelor HTML, în nici un caz pentru generarea de noi documente. Astfel, elemente ca BASEFONT, FONT, CENTER, S, STRIKE sau U sunt doar parte a documentelor tranzitionale, folosite pentru formatare.
- Documente *stricte*: acest tip nu include elementele prezente în documentele tranzitionale pentru compatibilitate cu versiunile anterioare de HTML.
- Documente tip *colectii de pagini*

Tipul de document se definește printr-o *definiție de tip* (concept preluat din SGML). Definiția de tip se specifica optional la începutul unui document HTML astfel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
http://www.w3c.org/TR/REC-html40/strict.dtd
```

Structura unui document HTML

Un document HTML conține date și marcaje (engl.tag). Marcajele conțin informația de formatare, iar datele reprezinta conținutul efectiv al documentului. Marcajele sunt interpretate de către programul navigator, ele determinand modul în care vor fi afisate datele. Marcajele pot fi: marcaje de început (engl.start tag) și marcaje de sfarsit (engl.end tag).

Un marcaj de început poate conține optional atribute cu valori și se scrie astfel:

```
<marcaj atribut = "valoare" atribut = "valoare" ...>
```

Ex:

Un marcaj de sfârșit se scrie astfel:

```
</marcaj>
```

Ex:

Teoretic, marcajele trebuie închise corect, exact ca parantezele dintr-o expresie matematică. În practică însă, deseori, marcajele de sfârșit se omit, ambiguitățile fiind rezolvate prin metode ad-hoc de programul navigator. În principiu, un document HTML conține un *antet* (engl.*header*) și un *corp* (engl.*body*) între marcajele <html> și </html>. html se numește și elementul rădăcină al documentului.

Elementele din corpul unui document HTML se clasifică în:

- Elemente de *nivel bloc* (engl.*block-level*). Ex: P, H, UL, OL, PRE, TABLE, FORM
- Elemente *incluse* (engl.*inline*). Ex: I, B, U, EM, STRONG, A, IMG

Diferența între ele se bazează pe trei coordonate: conținut, formatare și direccionalitate. Relativ la conținut, trebuie menționate următoarele aspecte:

- Un element de nivel bloc poate conține atât elemente incluse cât și elemente de nivel bloc.
- Un element inclus poate conține numai elemente incluse și date.
- Această distincție sugerează că elementele de nivel bloc pot crea structuri mai bogate decât elementele incluse.

În privința formătărilor, trebuie să se țină cont că:

- Elementele de nivel bloc sunt formate întotdeauna începând cu o linie nouă, în timp ce elementele incluse nu.
- Elementele incluse pot fi despartite pe mai multe linii (la fel cum textul unui paragraf se desparte în linii), în timp ce elementele de nivel bloc nu.

Direccionalitatea se referă la direcția în care este scris textul și se respectă regula următoare:

Elementele de nivel bloc mostenesc direccionalitatea de la elementul cuprinzător, în timp ce elementele incluse nu.

Un exemplu de document HTML este următorul :

```
<HTML>
<HEAD>
<TITLE> Un exemplu simplu de pagina WWW </TITLE>
</HEAD>
<BODY>
<H1> Amelia Badica despre HTML </H1>
Un document HTML conține:
<UL>
  <LI> marcaje - marcajele pot fi:</LI>
  <UL>
    <LI>marcaje de început</LI>
    <LI>marcaje de sfârșit</LI>
  </UL>
  <LI> date </LI>
</UL>
```


și este compus din:

```
<UL>
  <LI> un antet </LI>
  <LI> un corp </LI>
</UL>
</BODY>
</HTML>
```

Formatarea textului în HTML

Textul HTML poate fi *simplu* sau *structurat*.

- *Textul simplu* este format din *paragrafe* (p), separate de linii libere (br) sau *titluri* (engl.*headings*, h1, h2,..., h6). Textul poate avea stiluri (ex. bold b, italic i, etc), fonturi (font, basefont)
- *Textul structurat* cuprinde *liste* și *tabele*.
 - *Listele* sunt: *liste neordonate* (ul, li) cu elementele indicate cu simboluri nenumerice (engl.*bullet*), *liste ordonate* (ol, li) cu elementele numerotate, și *liste de definiții* (dl, dd) cu simbolurile elementelor definite de utilizator.
 - *Tabelele* (table) permit structurarea textului în linii și coloane. În principiu un tabel conține unul sau mai multe *corpuri de tabel* (engl.*table body*, tbody). Un corp de tabel conține una sau mai multe *linii* (engl.*row*, tr). O linie este formată din celule. Acestea pot fi *celule antet* (engl.*header cell*, th) sau *celule de date* (engl.*data cell*, td).
Pentru tabele există și alte elemente și atribute optionale care specifică titlul tabelului (caption), alinierea celulelor (align), stilul conturului (border), formatarea textului în celule, etc. Spre exemplu stilul (grosimea) conturului se specifică astfel: <table border="1">.

Legături HTML

Legăturile HTML se bazează pe folosirea de *ancore* (engl.*anchor*) cu ajutorul elementului a. Există două tipuri de ancere: *ancora sursă* și *ancora destinație*.

Ancorele sursă au forma

```
<a href="url">text</a>
```

și reprezintă originea unei legături. *url* reprezintă resursa destinație a legăturii. *text* este textul afișat în zona sensibilă a legăturii.

Ancorele destinație au forma

```
<a name="nume"></a>
```

și reprezintă destinația unei legături. Numele unei ancere destinație poate fi specificat într-o ancora sursă prin #*nume* pe post de *url*. Ancorele destinație se pot specifica și cu atributul *id* în cadrul elementului referit, în acest caz numai fiind nevoie să se folosească elementul a.

Documente HTML multifereastra

HTML 4.0 permite definirea de *colecții de pagini* (engl.*frameset*) pentru integrarea unei multimi de documente într-o aceeași fereastră principală. Fiecare document va fi vizualizat într-o porțiune a ferestrei principale.

Pentru definirea structurii multifereastra se folosește un document HTML special numit *tipar* (engl.*frameset document*) ce conține marcasele speciale frameset, frame și noframes (optional). Celelalte documente ale colecției sunt documente HTML obișnuite (engl.*frame document*).

Documentul tipar specifică documentele care se încarcă în fiecare fereastră. Dacă aceste

documente conțin legături spre alte documente, în cadrul acestor legături se poate specifica fereastra destinație prin atributul `target`. Numele ferestrelor se specifică cu atributul `name` în marcajul `frame`.

Specificarea dimensiunii ferestrelor în documentul tipar se face cu atributele `rows` și `cols`. Pentru fiecare fereastră se indică fie dimensiunea absolută (număr), fie procentul din spațiul total (procent) fie dimensiunea relativă (număr și *).

Se pot insera direct ferestre în cadrul unui document HTML obișnuit folosind marcajul `iframe`. Ele se numesc și *ferestre incluse* (engl.*inline frame*).

Imagini în documentele HTML

Cele mai răspândite formate de imagini întâlnite în WWW sunt Graphics Interchange Format – GIF, Joint Photographic Experts Group – JPEG și Portable Network Graphics – PNG. Ele se folosesc pentru stocarea imaginilor în forma compresată. De remarcat că HTML nu specifică tipul de format al imaginilor. Acesta este tratat exclusiv de programul navigator, imaginile fiind dpdv al HTML doar date externe.

Pentru includerea imaginilor în paginile WWW se folosește marcajul `IMG`. Atributele: `SRC` pentru specificarea fișierului care conține imaginea, `ALIGN` (cu valorile `left`, `right`, `top`, `middle` sau `bottom`) pentru specificarea tipului de aliniere, `ALT` pentru specificarea unei descrieri textuale a imaginii, `BORDER` pentru specificarea grosimii marginii imaginii, `HEIGHT` și `WIDTH` pentru specificarea înălțimii și lărgimii imaginii în număr de pixeli, etc.

Exemplu:

```
<IMG SRC="sunset.jpg" HEIGHT="300" WIDTH="400" ALT="Apus"/>
```

Pentru crearea de legături între imagini sau porțiuni de imagini și acțiuni sau resurse se folosesc *hartile de imagini* (engl.*image map*). Acțiunile sunt porțiuni de cod scripting iar resursele se specifică prin URL. Hartile de imagini se pot rezolva la server sau la client. Rezolvarea presupune găsirea resursei sau acțiunii corespunzătoare imaginii sau porțiunii de imagine care a fost selectată. Asocierea unei imagini cu o hartă se face cu atributul `USEMAP` pentru rezolvarea la client și cu atributul `ISMAP` pentru rezolvarea la server.

Formulare HTML și comunicare HTTP

Interacțiunea dintre CGI și formularele HTML se face astfel:

- *Formularele* (engl.*form*) furnizează paginilor WWW suport pentru interactivitate. Ele se specifică cu elementul `form`. Acesta acceptă atribute ca: *action* pentru a indica programul de prelucrare a datelor formularului, *method* pentru specificarea metodei HTTP folosită pentru transferul datelor către server (GET sau POST), *target* pentru a specifică fereastra care conține formularul, etc.
- În interiorul elementului `form` se introduc elemente pentru descrierea controalelor de interactivitate. Acestea pot fi:
 - `input`, un element general cu tipul descris prin atributul `type`. Acesta poate fi: `text` pentru o casetă de editare, `password` pentru o casetă de editare a unei parole, `checkbox` pentru o casetă de validare, `radio` pentru un buton de selecție exclusivă (engl.*radio button*), `submit` pentru un buton de trimitere a datelor către server, etc.
 - `select`, pentru definirea unui meniu. Opțiunile meniului se descriu cu elementele `option` și `optgroup`. Meniul poate fi cu selecție simplă sau multiplă (*multiple*)
 - `textarea`, pentru definirea unei câmp de editare multilinie.
- Pentru vizualizarea datelor trimise către server se va implementa formularul folosind metoda GET și se va vizualiza URL-ul generat în câmpul de adresă al navigatorului WWW.

5.2.2. Tehnologii DynamicHTML

Termenul de *programare pe partea de client* (engl.*client side programming*) se refera la executarea de programe la client in scopul cresterii gradului de interactivitate al paginilor WWW. Programele destinate a fi executate la client se pot transmite de la server către client fie in *format sursa*, fie in *format obiect*.

Termenul de *programare pe partea de server* (engl.*server-side programming*) se refera la executarea de programe la server in scopul de a genera date și/sau rapoarte care sunt trimise clientului sau mai general, pentru extinderea funcționalității unui server de WWW. Datele și rapoartele sunt trimise clientului de obicei in format HTML.

Termenul *Dynamic HTML* – DHTML nu se refera la o anumita versiune sau facilitate a HTML. El desemneaza acele facilitati din diversele variante sau extensii ale HTML care ajuta la crearea de conținut dinamic.

Cele mai populare elemente ale DHTML sunt:

- *Foile de stil* (engl.*Cascading Style Sheets* - CSS).
- *Scripting-ul la client* (elementul SCRIPT). Un exemplu este JavaScript. Se refera la incorporarea in cadrul unei pagini a unor programe in format sursa. Ele vor fi executate la client.
- *Obiectele* (elementul OBJECT). Se refera la incorporarea in cadrul unei pagini a unor programe in format obiect. Ele vor fi executate la client.
- *Modelul obiectelor document* (engl.*Document Object Model* – DOM). Acesta este liantul dintre elementele anterioare și limbajul de marcare HTML sau XML.
- Eventuale mecanisme particulare specifice programului navigator.

Se recomanda separarea *prezentării* (marginii, culori, fonturi) de *conținutul și structura* documentului (antet, pagini, paragrafe, titluri, sectiuni). Pentru aceasta se pot folosi *foi de stil* (engl.*Cascading Style Sheets* – CSS).

Prezentarea unui document este in general determinata de mai multi factori:

- Intențiile proiectantului paginii
- Reguli generale pe care trebuie sa le urmeze proiectantul
- Preferintele utilizatorului
- Limitările cauzate de terminalul pe care se vizualizeaza prezentarea

Din acest motiv apare necesitatea utilizării mai multor foi de stil – *cascaarea foilor de stil* (CSS). CSS specifica *stilul de prezentare* al unui document HTML. Exista trei modalitati de a mixa CSS și HTML: i) in cadrul unui element HTML; ii) in antetul unui document HTML; iii) intr-un fisier separat:

- Specificarea stilului intr-un element individual se face cu atributul STYLE:

```
<HTML>
<HEAD>
  <TITLE>
    Specificarea stilului in cadrul unui element HTML
  </TITLE>
</HEAD>
<BODY>
  <P STYLE = "font-size: 20pt">Text</P>
  <P STYLE = "font-size: 20pt; color: #0000FF">Text</P>
</BODY>
</HTML>
```

- Specificarea stilului in antet se face cu elementul STYLE:

```

<HTML>
<HEAD>
  <TITLE>Specificarea stilului in antet
</TITLE>
  <STYLE TYPE = "text/css">
    EM { background-color: #8000FF; color: white }
    H1 { font-family: Arial, sans-serif }
    P { font-size: 18pt }
    .blue { color: blue }
  </STYLE>
</HEAD>
<BODY>
  <H1 CLASS = "blue">Un antet</H1>
  <P>Un paragraf</P>
  <H1>Alt antet</H1>
  <P CLASS = "blue">Alt <EM>paragraf</EM></P>
</BODY>
</HTML>

```

- Foaie de stil intr-un fisier separat:

```

EM { background-color: #8000FF; color: white }
H1 { font-family: Arial,
  sans-serif }
P { font-size: 18pt }
.blue { color: blue }

```

- O foaie de stil pastrata intr-un fisier separat se leaga la un document HTML folosind elementul LINK.

```

<HTML>
<HEAD>
  <TITLE>
  Importul unei foi de stil
  <LINK REL="stylesheet"
  TYPE="text/css"
  HREF="css3.css"/>
</TITLE>
</HEAD>
<BODY>
  <H1 CLASS = "blue">Un antet</H1>
  <P>Un paragraf</P>
  <H1>Alt antet</H1>
  <P CLASS =
  "blue">Alt<EM>paragraf</EM></P>
</BODY>
</HTML>

```

Reguli, selectori și declarații

O foaie de stil este constituită dintr-o mulțime de *reguli* care se aplică unui document HTML în scopul generării unei prezentări a documentului.

O regulă de stil conține o *parte de condiții* (stanga) numită și *selector* și o *parte de acțiuni* (dreapta) numită și *declarație*. Exemplu: `P {color:green}`, unde:

- P este un selector
- {color:green} este o declarație. O declarație poate conține mai multe asignări
- color:green este o asignare
- color este o proprietate
- green este o valoare

Stilul unui document este o mulțime de asignări de valori unor variabile numite *proprietăți*. O proprietate exprimă o calitate sau caracteristică pe care o poate avea un element al unui document HTML.

Asignarea de valori proprietăților se bazează pe mecanismul de *mostenire cu suprascriere* (engl. *overriding inheritance*).

Un document HTML este structurat sub forma unui arbore de elemente cărora li se asociază proprietăți cu valori. O pereche proprietate – valoare se transmite prin mostenire de la nivelurile superioare în arbore către frunze. La întâlnirea unei noi valori pentru aceeași proprietate, noua valoare o va suprascrie pe cea veche, fiind considerată mai specifică.

Selectorii regulilor pot fi: tipurile elementelor, atributele elementelor (elementele CLASS și ID au fost introduse pentru a fi folosite cu CSS), contextul elementelor, respectiv informații externe documentului. Pentru detalii se va consulta specificația CSS1 (CSS de nivel 1).

Moștenirea și rezolvarea conflictelor în DHTML

În general unui document i se aplică mai multe foi de stil:

- Foaia de stil specifică documentului
- Foaia de stil implicată a programului navigator. Ideal, navigatorul ar trebui să permită setarea foii de stil implicite în funcție de preferințele utilizatorului, într-un mod independent de navigator
- O eventuală foaie de stil externă, specifică tipului de document

În cazul în care mai multe reguli sunt aplicabile unui element, selecția regulii aplicabile se face pe baza unei *strategii de rezolvare a conflictelor*. Conflictele sunt de două tipuri:

- Conflicte în cadrul aceleiași foi de stil
- Conflicte între regulile unor foi de stil diferite

Rezolvarea conflictelor se face astfel:

- Se determină toate regulile aplicabile din toate foile de stil.
- Se ordonează regulile selectate de pasul anterior astfel încât regulile marcate ca importante au prioritate.
- Se ordonează regulile selectate de pasul anterior în funcție de originea foii de stil: navigator, utilizator, proiectant (de la proiectant au precedența maximă). După acest pas vor rămâne doar reguli din aceeași foaie de stil.
- Se ordonează regulile selectate de pasul anterior în funcție de specificitate. O regulă este cu atât mai specifică cu cât selectorul ei exprimă o condiție mai complexă.
- Se ordonează regulile selectate de pasul anterior în funcție de ordinea în care au fost specificate. Regulile ce apar mai târziu în document au o prioritate mai mare. Regulile importate din foi de stil externe se consideră aflate înainte oricărui reguli prezente explicit în foaia de stil.

Cateva exemple de reguli in foile de stil sunt :

- Selector bazat pe tipul elementelor:
H1 {color: blue}
H1, H2: {color: blue}
 - Selector bazat pe clasa (atributul CLASS) elementelor:
.IMPORTANT {color: red}
 - Selector bazat pe ID-ul elementelor:
#COPYRIGHT {font-size: small}
 - Selector contextual:
LI P {margin-top: 0mm}
TABLE .SMALL P {font-size: small}
 - Selector bazati pe informatii externe documentului
A:LINK {color: green}
A:ACTIVE {color: red}
A:VISITED {color: blue}
P.INITIAL:FIRST-LETTER {font-size: 200%; float: left}
P.INITIAL:FIRST-LINE {text-transform: uppercase}
 - Combinarea tipurilor de selectori
TABLE P.INITIAL:FIRST-LETTER {font-size: 200%; float: left}
- Proprietatile din cadrul declaratiilor pot fi:
- Referitoare la fonturi
 - Referitoare la spatiere
 - Referitoare la imagini
 - Referitoare la culori și fundaluri

5.2.3. Miniaplicatii Java

O *miniaplicatie* (engl.*applet*) Java este un program Java care ruleaza sub controlul unui program client de navigare WWW. Miniaplicatia este transferata de la server către client sub forma de cod obiect (*class* sau *jar*).

Conditia ca o miniaplicatie sa poata fi executata pe calculatorul clientului este ca programul de navigare sa dispuna de un subprogram de interpretare a codului binar al masinii virtuale Java. Un astfel de program navigator se numeste *Java-enabled*.

Restrictii:

- i) o miniaplicatie nu se poate atinge (in citire sau scriere) de discul calculatorului client pe care ruleaza; din acest motiv se spune metaforic ca ruleaza *inside the sandbox*;
- ii) lansarea in executie a unei miniaplicatii poate dura deoarece descarcarea fisierelor *class* și apoi incarcarea lor sub controlul programului client de navigare poate consuma un timp semnificativ de mare.

O miniaplicatie se include intr-o pagina WWW cu elementele APPLET sau OBJECT. Initial s-a folosit elementul APPLET, deoarece singurele programe executabile care se puteau include in paginile WWW sub forma de cod obiect erau miniaplicatiile Java. Exemplu:

```
<APPLET CODE="Miniaplicatie" CODEBASE="." WIDTH="300" HEIGHT="200">
</APPLET>
```

Ulterior s-a introdus elementul OBJECT pentru a se permite includerea și a altor aplicatii sub forma de cod obiect in paginile WWW. Exemplu:

```
<OBJECT WIDTH="500" HEIGHT="300"> <PARAM NAME="code" VALUE=
"Miniaplicatie"/>
```

```
<PARAM NAME="codebase" VALUE = "."/>
```

```
</OBJECT>
```

Construirea unei miniaplicatii Java presupune urmatoarele:

- O miniaplicatie Java este o aplicatie grafica care foloseste bibliotecile Java pentru construirea de intrft grafice AWT sau *Swing*.
- O miniaplicatie nu are constructor deoarece este creata de programul de navigare sub controlul caruia ruleaza. Ea se construiesc prin extinderea clasei Applet din AWT sau JApplet din Swing și redefinirea urmatoarelor metode (cel puțin `init()` trebuie redefinita):
 - `init()`, se executa o singura data la incarcarea miniaplicatiei.
 - `start()`, se executa după `init()` și ori de cate ori este reafisata pagina care conține miniaplicatia
 - `stop()`, se executa cand pagina ce conține miniaplicatia nu mai este afisata și imediat înainte de `destroy()`
 - `destroy()`, se executa cand miniaplicatia se inchide și trebuie descarcata din memorie
- Operatiile din cadrul redefinirii metodei `init()` sunt asemenea cu cele din cadrul unui constructor al unei aplicatii grafice obisnuite. Spre exemplu, aici se initializeaza controalele grafice membre ale interfetei miniaplicatiei și se adauga la miniaplicatie. Din acest motiv orice miniaplicatie poate fi rulata și ca o aplicatie obisnuita dacă se definește o metoda `main()` in care se creeaza un obiect miniaplicatie și apoi se executa operatiile din `init()`.

Interfete grafice in Java

Se spune ca AWT (`java.awt`) este de *categorie grea* (engl.*heavyweight*), iar Swing (`javax.swing`) este de *categorie usoara* (engl.*lightweight*). AWT se bazeaza pe obiectele grafice ale platformei gazda in timp ce Swing redeseneaza toate controalele grafice. Astfel AWT este mai rapida, dar dependenta de platforma (la aspect) in timp ce Swing este mai lenta, dar independenta de platforma.

O interfata grafica este compusa din *componente*. Componentele pot fi amplasate in *containere*, care sunt la randul lor componente. Astfel se pot construi interfete grafice complexe. Modelul de programare folosit la programarea interfetelor grafice se numeste *programare orientata pe evenimente* (engl.*event-driven programming*). Componentele genereaza evenimente la actiunile utilizatorului. La componente se inregistreaza niste obiecte speciale numite *obiecte ascultator* (engl.*listener*) interesate sa primeasca aceste evenimente.

Obiectele ascultator primesc evenimentele sub forma unor apeluri de metode. Din acest motiv ele trebuie sa implementeze anumite interfete astfel incat sa permita aceste apeluri. Aceste interfete sunt subclase ale clasei de baza `EventListener`.

În momentul în care un obiect ascultător recepționează un eveniment, în apelul corespunzător evenimentului se transmit și informații referitoare la obiectul grafic care a transmis evenimentul.

5.2.4. Programare la client folosind *scripting*

Prin limbaj de *scripting* se înțelege un limbaj de programare interpretat. Soluția de interpretare permite transmiterea programelor în cod sursă, cu condiția existenței la destinație a unui interpretor pentru executia lor. Exemple de limbaje de *scripting*: dBase, Visual Basic, Perl, Unix shell, Python, Tcl, Ruby, etc. Un program scris într-un limbaj de *scripting* se numește *script*.

În WWW, *scripting*-ul se folosește atât la client cât și la server. Executia unui *script* la server poate produce conținut ce este transmis prin HTTP clientului (de exemplu prin CGI). Executia unui *script* la client produce conținut dinamic. *Scripting*-ul la client este parte a tehnologiei *Dynamic HTML – DHTML*.

Deosebirea esențială dintre un limbaj de programare compilat și un limbaj de *scripting* interpretat este *momentul legării* (engl.*binding*) - momentul în care devin cunoscute atributele unei variabile. La limbajele compilate momentul legării este cel al traducerii, iar la cele interpretate este cel al executiei. Amanarea legării atributelor conduce la flexibilitate a executiei în dauna eficienței.

JavaScript este un *limbaj de scripting orientat pe obiect* inventat de Netscape. Spre deosebire de limbajele orientate pe obiect bazate pe clase, cum sunt C++ și Java, JavaScript este un *limbaj orientat pe obiect bazat pe prototipuri*.

JavaScript este un limbaj *extensibil*. Există o *componentă de bază*, numită *Core JavaScript*, care conține elementele de bază ale limbajului (cuvinte cheie, operatori, enunțuri, structuri de control, modelul obiectual) și o multitudine de obiecte de bază (ex.*Array, Date, Math*). La ea se pot adăuga extensii sub forma unor obiecte suplimentare. Exemple sunt *Client-side JavaScript* care conține în plus obiecte pentru controlul programului navigator și DOM și *Server-side JavaScript* care conține obiecte suport pentru partea de server (de ex. pentru comunicarea cu o bază de date relatională). În continuare prin JavaScript vom subînțelege partea client a JavaScript.

Navigatoarele WWW interpretează scripturile JavaScript din paginile HTML. Ele citesc pagina, interpretează marcăjele și o afișează, executând în același timp scripturile JavaScript pe măsura întâlnirii lor în cadrul paginii. Rezultatul acestui proces de interpretare/execuție este vizualizat de utilizator în fereastra navigatorului.

Între JavaScript și Java există și asemănări și deosebiri fundamentale. Asemănările se referă la sintaxa enunțurilor și a structurilor de control. Deosebirile: Java are *legare la compilare* (engl.*statically typed*), este *puternic tipizat* (engl.*strongly-typed*) și folosește un *model obiectual bazat pe clasă*. JavaScript are *legare la execuție* (engl.*dynamically typed*), este mult mai permisiv în ceea ce privește declarațiile (engl.*loosely-typed*) și folosește un *model obiectual bazat pe prototipuri*.

Utilizarea JavaScript într-o pagină HTML presupune folosirea elementului *SCRIPT*:

```
<SCRIPT> Enunțuri JavaScript... </SCRIPT>
```

Specificarea limbajului și versiunii de *scripting* se face cu atributul *LANGUAGE*:

```
<SCRIPT LANGUAGE="JavaScript1.3">
```

Specificarea codului JavaScript se poate face direct în elementul *SCRIPT*:

```
<HTML>
```

```
<HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript1.3">
```

```
<!-- Ascunde scriptul pentru navigatoarele ce nu știu Javascript.
```

```
document.write("Salut !");
```

```
// Sfârșitul scriptului. -->
```



```
</SCRIPT>
</HEAD>
</HTML>
```

Specificarea codului JavaScript se poate face și într-un fisier/URL separat:

```
<HTML>
<HEAD>
  <SCRIPT SRC="js1.js"></SCRIPT>
</HEAD>
</HTML>
```

Utilizarea *entitatilor JavaScript* permite specificarea de expresii JavaScript ca valori ale atributelor elementelor HTML (nu merge la IE !?). O entitate JavaScript are sintaxa `&{expresie JavaScript}`; Un exemplu de folosire:

```
<SCRIPT LANGUAGE="JavaScript">
  var lungime = 25;
</SCRIPT>
<HR WIDTH="{lungime*2};%" ALIGN="left"/>
```

Sirurile de caractere in cadrul unui literal JavaScript se pun intre apostroafe (merge și \" in IE):

```
document.writeln("<HR ALIGN='left' WIDTH=" + lungime*2 + "%>")
```

Aplicatiile JavaScript sunt *orientate pe evenimente* (engl.*event-driven*). Un eveniment este de obicei rezultatul unei actiuni a utilizatorului. Un script poate reactiona la evenimente prin intermediul unui secvente de cod de tratare a evenimentului (engl.*event handler*). Un event handler este o secventa de cod JavaScript, de obicei un apel de funcție JavaScript.

Dacă un eveniment se aplica unui element HTML, atunci se poate specifica un event handler in cadrul elementului. Evenimentul se aplica obiectului JavaScript creat de element.

```
<element onEveniment="cod JavaScript">
```

Exemplu:

```
<HEAD>
<SCRIPT>
  funcțion calcul(f) {
    f.rez.value = eval(f.expr.value)
  }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
  Introduceți o expresie:
  <INPUT TYPE="text" NAME="expr" SIZE=15/>
  <INPUT TYPE="button" VALUE="Calculeaza" onClick="calcul(this.form)"/><BR/>
  Rezultat:
  <INPUT TYPE="text" NAME="rez" SIZE=15/>
</FORM>
</BODY>
```

Obiectul `this` desemneaza obiectul buton, iar `this.form` formularul care conține butonul. Funcția `calcul()` primește ca argument un formular, preia expresia de calculat din câmpul `expr`, o evaluează cu `eval()` și depune rezultatul în câmpul `rez`. Evenimentul tratat este *Click* pe buton.

Core JavaScript

Variabilele se declară la inițializare sau optional cu `var`. Pot fi *globale* sau *locale* (definite în corpul unei funcții). Cele locale se declară obligatoriu cu `var`.

Există tipuri primitive și tipuri compuse. *Tipurile primitive* sunt: numerele (întregi și reali), valorile logice, sirurile de caractere, valoarea `null` și valoarea `undefined`. *Tipurile compuse* sunt *vectorii* și *obiectele*.

Pentru specificarea *constantelor* (numite și *valori*) se folosesc *literalii*. Există literalii pentru specificarea numerelor (2, 3.14), valorilor logice (`true` și `false`), vectorilor (`[1,2,3,"aaa"]`) și obiectelor (`{camp_1:"a",camp_2:3}`).

Structurile de control sunt cele din Java: secvențierea, `if-else`, `switch`, `for`, `while`, `do-while`, `label`, `break` și `continue`. Există în plus *structurile pentru manipularea obiectelor*: `for-in` și `with`. *Comentariile* sunt ca în Java.

Caramizile de bază ale JavaScript sunt *funcțiile*. Ele se declară astfel:

function nume(argumente) { corpul funcției }

Funcțiile pot întoarce optional o valoare cu `return` și pot fi recursive. Argumentele funcțiilor sunt disponibile din vectorul `arguments`. Există o multime de funcții predefinite (vezi documentația de la Netscape).

Funcțiile și variabilele globale pot fi referite și din alte ferestre dacă se prefixează cu numele ferestrei în care au fost definite.

Relativ la modelul obiectual din JavaScript, se menționează următoarele aspecte:

- Construirea unui obiect presupune:
- Inițializarea cu un literal:

```
nume = {prop1:val1, prop2:value2, ..., propn:valn};
```

- Definierea unei funcții constructor:

```
funcția TipObiect(argumente) {  
  this.prop1 = val1;  
  ...  
}
```

```
obiect = new TipObiect(argumente);
```

- Proprietățile unui obiect se pot accesa cu `obiect.prop` sau cu `obiect["prop"]`.
- Obiecte JavaScript predefinite: `Array`, `Boolean`, `Date`, `Funcția`, `Math`, `Number`, `RegExp`, `String`.
- *Ierarhii de obiecte* în JavaScript: dacă se dorește crearea unui obiect care mostenește de la un alt obiect se setează valoarea `prototype` a sa la obiectul de la care se mostenește. Noțiunea de *clasa de bază* din limbajele bazate pe clase a fost înlocuită cu noțiunea de *obiect prototip*.

Concluzii:

- Pe cât posibil încercați să separați partea de conținut de partea de prezentare din pagini. Pentru aceasta identificați întâi unitățile de conținut pe care trebuie să le conțină pagina și abia apoi puneți-vă problema modului de prezentare al acestora.
- Aveți în vedere că descărcarea unor programe în cod obiect și rularea lor în programul navigator poate duce la încetinirea încărcării paginii.
- Folosiți scriptingul la client numai atunci când este absolut necesar. Scriptingul nu face să crească dependentă paginilor pe care le scrieți de programul navigator.
- Atunci când doriți să creați pagini complexe și folosirea scriptingului la client este absolut

necesara, incercati sa identificati cu clientul dumneavoastra navigatoarele carora le sunt adresate paginile. Un pas important va fi apoi testarea navigatorului inainte de afisarea paginilor sau a anumitor parti specifice din pagini.

5.3. PROGRAMARE PE PARTEA DE SERVER

5.3.1. Servere WWW

Un *server WWW* este un program cu rol de server care este capabil sa raspunda la cereri HTTP. Modul de funcționare al unui server WWW este foarte important pentru infrastructura și aplicatiile WWW, desi in acelasi timp este ascuns utilizatorilor uzuali ai WWW. Probleme importante referitoare la serverele WWW sunt:

- Performanta
- Configurarea și administrarea
- Extensia și programarea

Un exemplu de server WWW foarte folosit și disponibil liber in domeniul public este Apache. Un sondaj din 1998 a aratat ca 51.9% din toate serverele active de WWW erau servere Apache.

Relativ la performantele serverelor WWW, trebuie precizate următoarele aspecte:

- La nivelul anului 1999 se mentioneaza in literatura doua suite (pachete de programe) standard (engl.*benchmark*) pentru testarea performantelor serverelor WWW:
 - WebStone, dezvoltata de Silicon Graphics, Inc. – SGI
 - SPECweb96, dezvoltata de Standard Performance Evaluation Corporation – SPEC
- WebStone este o suita configurabila de teste de performanta, destinata a fi utilizata de administratorii WWW. Configurabilitatea este utila pentru a proiecta teste specifice pentru diverse sabloane de acces la servere.
- SPECweb96 este o suita fixa de teste de performanta. Sabloanele de test au fost obținute analizand cele mai frecvente moduri de utilizare ale serverelor WWW.
 - Optiunile de configurare ale unui server WWW se refera in general la:
 - Modul in care se va executa serverul pe masina care il gazduieste.
 - Dacă serverul deserveste mai multe masini gazda și in caz afirmativ modul in care realizeaza acest lucru. Dacă serverul deserveste mai multe masini gazda atunci acestea se numesc *gazde virtuale* (engl.*virtual host*).
 - Modul de lucru: *server de origine* sau *proxy*.

Executia serverelor WWW poate fi de două tipuri:

- Executie la cerere
 - Este o metoda veche. Presupune existenta unui *superserver* care asculta toate cererile și pentru fiecare in parte activeaza și executa serverul corespunzator. In UNIX acest superserver se numeste *inetd*. Aceasta metoda are marele dezavantaj ca necesita prea multe resurse sistem de fiecare data cand se incarca serverul in memorie și se creaza procesul aferent.
- Executie permanenta
 - Serverul este pornit manual de utilizator sau automat la pornirea sistemului și se executa in permanenta. După pornire serverul asculta cererile HTTP pe portul pe care a fost configurat. Pornirea manuala este recomandata intr-un mediu de dezvoltare a unei aplicatii WWW. Pornirea automata este recomandata după ce aplicatia WWW a fost instalata (engl.*deployed*).
 - Serverul poate fi pornit de la linia de comanda sau instalat ca serviciu al sistemului de operare.

5.3.2. Gazde virtuale

Un server WWW poate deservi una sau mai multe masini gazda. In al doilea caz masinile gazda deservite se numesc *gazde virtuale*. Gazdele virtuale usureaza activitatea de administrare a serverului deoarece: exista o singura instanta a serverului, exista o singura configuratie a serverului, se monitorizeaza executia unui singur server. Exista doua tipuri de gazde virtuale: *gazde virtuale IP* și *gazde virtuale non-IP*.

Gazdele virtuale IP:

- Fiecare gazda virtuala este o gazda IP avand o adresa de IP distincta care apare ca intrare in DNS.
- Se asigneaza toate adresele de IP ale gazdelor virtuale deservite masinii pe care ruleaza serverul WWW.
- Metoda este simpla dar are doua dezavantaje: i) este nevoie de o adresa de IP distincta pentru fiecare gazda virtuala; ii) asignarea mai multor adrese de IP unei singure masini poate crea probleme retelei.

Gazdele virtuale non-IP:

- Se bazeaza pe un suport special oferit de protocolul HTTP. Acest suport pentru deservirea gazdelor virtuale non-IP este disponibil numai in HTTP/1.1.
- HTTP/1.1 a introdus in cadrul unei cereri HTTP antetul special HOST. Acest camp antet este obligatoriu. El conține numele gazdei careia ii este adresata cererea.
- Configurarea gazdelor virtuale deservite se face analog cu cazul anterior. Singura diferenta este ca intrarile DNS ale gazdelor virtuale vor conține acum aceeasi adresa de IP.
- Metoda are doua avantaje: i) este nevoie de o singura adresa de IP pentru deservirea tuturor gazdelor virtuale; ii) crearea unei noi gazde virtuale non-IP se poate face mai usor decat in cazul gazdelor virtuale IP. Metoda are un mare dezavantaj: se poate aplica numai pentru versiunile HTTP ulterioare HTTP/1.1.

Serverele WWW pot fi:

➤ **Servere de origine**

- Majoritatea serverelor de WWW sunt configurate ca *servere de origine*. In acest caz cererile HTTP sunt tratate local de server.
- O problema importanta este localizarea resurselor, cu alte cuvinte maparea URL-ului intr-o adresa a unei resurse din cadrul sistemului local de fisiere. Resursele sunt de doua tipuri: *statice* (stocate fizic) și *dinamice* (generate de programe externe).
- Spatiul WWW al unui utilizator specific se refera in URL prin caracterul ~ urmat de numele utilizatorului. Spatiul WWW al utilizatorilor se poate organiza in doua moduri:
 - Crearea unui spatiu WWW central pentru toti utilizatorii. Unui utilizator ii va revni un director in acest spatiu: `webhome/name/`.
 - Crearea unui director special pentru fiecare utilizator in cadrul directorului personal: `/home/name/WWW/`.

➤ **Servere proxy**

- Un server proxy accepta cereri pentru resurse și fie le rezolva din memoria *cache* locala, fie prin inaintarea cererii către serverul de origine.
- Un server proxy actioneaza astfel ca un intermediar intre client și serverul de origine. Scopul intermediarii poate fi filtrarea cererilor sau trecerea de la un protocol nesigur la un protocol sigur, de exemplu HTTPS.

5.3.3. Tehnologiile SSI și CGI

SSI (engl. *Server Side Include*) este o tehnologie ce permite includerea de informații într-o pagina WWW înainte de trimiterea paginii la client.

O pagina care folosește SSI conține instrucțiuni speciale. Aceste instrucțiuni sunt interpretate de server ori de câte ori pagina este cerută de un client. Instrucțiunile pot specifica: includerea unor documente în pagina curentă (de exemplu un *header* sau un *footer*), a datei curente, a valorii unui contor de acces, etc.

Avantajul SSI față de CGI este că este o tehnologie mult mai simplă și că nu implică apelul nici unui program extern.

Nu există un standard de SSI și de aceea fiecare server are o sintaxă și funcționalitate proprie pentru SSI. Pentru serverul Apache instrucțiunile SSI sunt de forma unor comentarii HTML/XML cu structura următoare:

```
<!-- #element atribut1=valoare1 atribut2=valoare2 ... -->
```

Un exemplu este următorul:

```
<!--#include virtual="/header.html" -->
```

Serverul Apache permite folosirea și a unor facilități SSI avansate desemnate prin acronimul XSSI. Un exemplu este posibilitatea definirii unui flux de control ca în programarea procedurală prin comenzile *if*, *elif*, *else* și *endif*.

Aplicațiile de comerț electronic au o parte semnificativă pe partea de server. Pentru a descrie această parte se folosește frecvent termenul de *aplicație WWW* = extensia dinamică a unui server WWW. Aplicațiile WWW sunt în general de două tipuri:

- *Aplicații orientate pe prezentare*. Conțin pagini WWW interactive și sunt capabile să genereze conținut dinamic ca răspuns la cererile clienților.
- *Aplicații orientate pe serviciu*. Implementează un punct terminal pentru un serviciu WWW. *Serviciu WWW* = un serviciu oferit de o aplicație altor aplicații, prin intermediul platformei WWW.

Aplicațiile WWW conțin *componente WWW*. Componentele WWW sunt caramizile pe baza cărora se poate extinde dinamic funcționalitatea unui server WWW. În tehnologia Java, componentele WWW sunt miniservere sau pagini JSP.

Componentele WWW sunt suportate de serviciile unei platforme speciale de execuție numită *container WWW*. Containerul furnizează servicii ca: dispunerizarea cererilor, securitate, concurența, gestiunea ciclului de viață, etc.

O aplicație WWW constă în general din: componente WWW, fișiere de resurse statice și clase/biblioteci de ajutor (engl. *helper*).

Funcționalitatea standard oferită de un server WWW este insuficientă pentru programarea aplicațiilor de comerț electronic. O aplicație de comerț electronic are o arhitectură multistrat, numărul de straturi fiind de obicei minim 3: client, server WWW, server de baze de date. Pe lângă serverul WWW mai există obiectele din domeniul problemei (engl. *business objects*). Împreună ele formează stratul intermediar al unei arhitecturi tipice pentru o aplicație de comerț electronic. În consecință se pune problema extinderii funcționalității unui server de WWW. O variantă este folosirea interfeței CGI, însă ea prezintă o serie de dezavantaje:

- Pentru fiecare cerere HTTP trebuie startat un proces separat.
- Interfața dintre serverul WWW și scriptul CGI este nesatisfăcătoare.

O abordare elegantă o constituie o soluție Java pentru extinderea funcționalității unui server WWW, și anume miniserverele Java (engl. *Java servlet*). Această soluție are următoarele avantaje:

- Portabilitate

- Miniserverele sunt rezidente și starea este persistentă între cereri succesive
- Beneficierea de avantajele tehnologiei Java: orientare pe obiect, modelul de securitate Java, integrarea relativ ușoară cu tehnologii ca RMI și CORBA

CGI (engl. Common Gateway Interface) definește o interfață pentru comunicarea dintre un server de informații (cum este cazul unui server WWW) și un program de aplicație.

CGI este o interfață independentă de limbaj. Este posibil să se implementeze o aplicație CGI (numită și *script CGI*) în orice limbaj care suportă comunicarea standard între procese prin intermediul intrării și ieșirii standard și a variabilelor de mediu. Pentru scripturile CGI se folosește deseori unul dintre limbajele: Perl, Python sau Tcl. Se pot folosi însă și limbaje gen C/C++.

Procesul de comunicare decurge astfel:

- După primirea cererii HTTP și înainte pornirii scriptului CGI, serverul inițializează o mulțime de variabile de mediu (engl. *environment variables*). Aceste variabile vor fi mostenite de procesul corespunzător lansării scriptului CGI. Există variabile independente de cerere și variabile dependente de cerere. Pe lângă aceste variabile de mediu, dacă protocolul este HTTP, se creează câte o variabilă de mediu ce începe cu prefixul HTTP pentru fiecare linie din antetul cererii (de exemplu HTTP_ACCEPT).
- Dacă cererea conține informații adiționale după antetul cererii, atunci informația este trimisă scripului CGI la intrarea standard. Se trimite un număr de octeți egal cu valoarea variabilei de mediu CONTENT_LENGTH.
- Scriptul generează datele de ieșire la ieșirea standard. Pentru generarea răspunsului către client, serverul fie interpretează și prelucrează aceste date, fie le înaintează nealterate. El indică acest lucru serverului prin antete speciale.

Funcționarea miniserverelor Java presupune următorii pași:

- Un program client emite o cerere HTTP către un server WWW.
- Serverul interpretează cererea și execută o secvență de program careia îi transmite parametrii cererii.
- Programul apelat interpretează parametrii cererii și execută o porțiune de cod care generează dinamic o pagină HTML.

Prelucrările executate de miniserver pot duce la:

- Generarea unei pagini WWW statice.
- Generarea unei pagini WWW modificată prin inserarea unui conținut dinamic.
- Configurarea unei pagini WWW pe baza parametrilor cererii HTTP. Parametrii sunt preluați de la utilizator printr-un formular HTML.

În general miniserverele sunt potrivite pentru aplicațiile orientate pe serviciu sau pentru controlul aplicațiilor orientate pe prezentare.

5.3.4. Miniservere Java

La fel ca și o miniaplicație, un miniserver nu conține o funcție `main()`. El va fi invocat de către containerul de miniserveri la recepționarea unei cereri HTTP de către serverul WWW. Pentru aceasta un miniserver trebuie să implementeze o interfață `javax.servlet.HttpServlet`

Dezvoltarea unui miniserver presupune extinderea clasei `javax.servlet.HttpServlet` și redefinirea metodelor sale. Interfața cuprinde metodele `service()`, `init()` și `destroy()`.

La executarea unei cereri către un miniserver au loc următoarele: i) dacă miniserverul nu există atunci containerul de miniserveri încarcă clasa corespunzătoare miniserverului, creează o instanță a acestei clase și apelează metoda `init()`; ii) se apelează metoda `service()`. În funcție de tipul cererii, ea va apela o metodă `doYYY()` unde `YYY` identifică cererea HTTP, de exemplu

doGet() sau doPost().

Cand containerul trebuie sa descarce miniserverul din memorie el va apela metoda destroy().

Metodele init() și destroy() se refera la ciclul de viata al miniserverului (la fel ca la miniaplicatii). Aceste operatii se executa mult mai rar decat service().

Prelucrarile din cadrul unui miniserver se fac astfel:

- In cadrul metodelor init() și destroy() se implementeaza acele prelucrari care asigura persistenta informației stocate de miniserver.
- In metodele service() sau doYYY() se implementeaza prelucrarile legate de tratarea cererilor primite de la client. Acestea presupun de obicei urmatoarele:
- Setarea tipului conținutului in obiectul raspuns HTTP, de tip javax.servlet.http.HttpServletResponse:

```
raspuns.setContentType("text/html");
```

- Obținerea unui flux de iesire la nivel de octet (OutputStream obținut din raspuns cu getOutputStream()) sau la nivel de caracter (PrintWriter obținut din raspuns cu getWriter()):

```
PrintWriter iesire = raspuns.getWriter();
```

- Obținerea valorilor parametrilor cererii setati de client in formularul HTML, din obiectul cerere de tip javax.servlet.http.HttpServletResponse, cu getParameter(). Dacă numele parametrilor este necunoscut atunci se poate obține lista numelor parametrilor cu getParameterNames().
- Construirea raspunsului prin scrierea in fluxul de iesire a unor enunturi HTML.
- Pentru aflarea de informații de configurare se foloseste metoda getServletConfig() care intoarce un obiect ServletConfig. Din el se pot extrage parametrii de initializare ai miniserverului cu getInitParameter() și getInitParameterNames().

Obiecte partajate:

- Componentele WWW in general și miniserverele in particular folosesc de obicei și alte obiecte pentru a-și indeplini sarcinile. Astfel: i) pot folosi obiecte private; ii) pot folosi obiecte publice – atribute ale unui domeniu standard; iii) pot accesa baze de date; iv) pot accesa alte resurse WWW.
- Componentele WWW cooperante pot partaja informații sub forma unor obiecte definite ca atribute ale unui domeniu standard: *context WWW (aplicatie), sesiune, cerere HTTP* și respectiv *pagina JSP*.
- Pentru reprezentarea acestor patru domenii, in pachetul javax.servlet sunt definite patru clase: ServletContext, HttpSession, ServletRequest și JspContext.
- Accesul la aceste obiecte se face prin metode de tip [get|set]Attribute.

Fire multiple in miniservere:

- Containerul de miniserveri utilizeaza fire de executie separate pentru tratarea cererilor. Pentru aceasta containerul are o rezerva (engl.*pool*) de fire de executie. Fiecarei cereri i se aloca un fir.
- Deoarece este teoretic posibil sa se execute cereri simultane pentru un acelasi miniserver, se pune problema gestionarii corecte a resurselor miniserverului ce sunt partajate de aceste cereri multiple. Resursele partajate pot fi: atribute ale domeniilor standard, variabile membre ale clasei miniserverului, fisiere, baze de date, conexiuni de retea, etc.
- Din acest motiv controlul accesului la resursele partajate se va realiza folosind tehnici de sincronizare.

- Variante:
 - Cea mai simpla solutie este ca metoda `service()` sa se implementeze cu `synchronized`.
 - Dacă însă secțiunea critică din cadrul acestei funcții se execută condiționat, atunci este mai eficient să se folosească `synchronized` doar pentru porțiunea de cod corespunzătoare acelei secțiuni critice.
 - Se poate crea o clasă pentru accesul la resursa partajată, metodele de acces fiind declarate `synchronized`.

Gestiunea sesiunii de lucru:

- O problemă majoră a HTTP este că nu permite păstrarea stării comunicării dintre un client și un server WWW. Pentru a înlătura această problemă s-au propus diverse metode:
- *Folosirea câmpurilor ascunse.* Această metodă presupune transmiterea repetată de la server către client și apoi de la client către server a istoricului interacțiunii sub forma unei mulțimi de câmpuri ascunse într-un formular HTML. Un exemplu de câmp ascuns este `<INPUT TYPE="hidden" name="ascuns1" value="element1">`. De fiecare dată când utilizatorul selectează un nou element serverul primește o pereche nume-valoare de forma `element_selectat="..."`, după care serverul creează un nou câmp ascuns pentru acest element și îl pasează în formularul răspuns, etc
- *Rescrierea URL-ului.* Această metodă constă în adăugarea de informații suplimentare la un URL pentru a identifica în mod unic o sesiune.
- *Folosirea autentificării HTTP.* Această metodă presupune identificarea și autentificarea utilizatorului la primul sau acces asupra serverului. Ulterior serverul va identifica utilizatorul din antetele HTTP.
- *Folosirea cookie-urilor.* Un cookie este o informație pe care un server WWW o poate stoca pe calculatorul unui client. Ulterior, această informație va fi trimisă serverului la fiecare cerere a clientului. Serverul poate folosi această informație pentru gestiunea sesiunii.

Gestiunea sesiunii de lucru folosind miniservere:

- Un *cookie* se creează și se adaugă la răspunsul HTTP înainte de setarea tipului conținutului.

```
Cookie c = new Cookie("element_selectat", "carte1");
raspuns.addCookie(c);
```
- Serverul va determina ce *cookie*-uri există în cererea HTTP cu `getCookies()`.
- O *sesiune* constă în una sau mai multe cereri HTTP către un server WWW de-a lungul unei perioade de timp.
- Implementarea unei sesiuni folosește clasa `HttpSession`. Un obiect sesiune este păstrat pe server și el stochează informațiile despre un client de-a lungul unei sesiuni a clientului respectiv.
- Clasa `HttpSession` folosește clasa `Cookie` pentru stocarea identificatoului sesiunii la client sub forma unui *cookie*. Sesiunea este păstrată un interval maxim de timp între două cereri succesive ale clientului.
- Informația se păstrează în obiectul sesiune sub forma unor perechi variabilă-valoare. Pentru accesul la aceste perechi se folosesc metodele `setAttribute()`, `removeAttribute()`, `getAttribute()`, `getAttributeNames()`;

Dezvoltarea unei aplicații care folosește miniservere presupune următoarele:

- Se compilează sursele *java* pentru a obține fișierele *class*.
- Se creează un subdirector în directorul *webapps*. Acest subdirector conține pagina rădăcină a aplicației și se numește *context*.
- Se creează un subdirector *WEB-INF* care va conține:
 - un fișier de configurare a aplicației *web.xml*

- un subdirector *classes* unde se depun fisierele *class*, pe subdirectoare corespunzatoare structurii de pachete a programului *java*
- Apelul miniserverului se poate face direct, dintr-un formular sau dintr-o legatura. In toate cazurile se va specifica un URL de forma:

URLGazda/context/servlet/fisier_class

Un exemplu este:

http://localhost/Miniserver1/servlet/Miniserver1

- Un miniserver poate fi repornit fie prin repornirea containerului de miniservere, fie doar prin repornirea lui individuala.

JSP (engl. *Java Server Pages*) este o tehnologie pentru generarea de pagini dinamice bazata pe ideea mixarii de cod Java cu cod HTML in paginile WWW. Este recomandata pentru aplicatiile WWW orientate pe prezentare.

O alta tehnologie de pagini dinamice foarte folosita la ora actuala este PHP. PHP este un limbaj de scripting pentru partea de server, inspirat din C. Codul PHP poate fi incorporat in paginile WWW.

JSP funcționeaza peste o arhitectura de miniservere. La incarcarea unui JSP, se genereaza automat codul *java* pentru miniserverul corespunzator și apoi acesta este compilat și incarcat in containerul de miniserveri. Acest proces se repeta ori de cate ori codul JSP este modificat.

Elemente de baza ale JSP:

- In JSP se pot referi niste obiecte predefinite: `HttpServletRequest request`, `HttpServletResponse response`, `jsp.PageContext pageContext`, `http.HttpSession session`, `ServletContext application`, `jsp.JspWriter out`, `ServerConfig config`, `java.lang.Object page`.
- Exista trei tipuri de elemente de scripting in JSP:
- *Expresii* de forma `<%= expresie %>`
- *Miniscripturi* (engl. *scriptlet*) de forma `<% cod Java %>`
- *Declaratii* de forma `<%! Cod %>`
- Expresiile se utilizeaza pentru a insera valori direct in pagina generata. De exemplu:
Current time: `<%= new java.util.Date() %>`
- Miniscripturile reprezinta un cod Java mai complex decat niste expresii simple.
`<% String queryData = request.getQueryString();
out.println("Date GET:" + queryData); %>`
- Declaratiile permit inserarea de metode sau campuri in codul *java* al corpului clasei miniserver, in afara metodei `_jspService()`.
`<%! private int jj =10; %>`

5.4. SERVERE DE BAZE DE DATE

Bazele de date relationale bazate pe SQL sunt cele mai raspandite sisteme comerciale de baze de date. Prin *baza de date relationala* vom intelege o multime de relatii (tabele) accesate prin limbajul de specializat SQL.

Un *tabel* sau *relatie* poate fi gandit ca o reprezentare bidimensionala a unei colectii de date, structurata sub forma unei multimi de linii și coloane. O *linie* corespunde unei *inregistrari* și o *coloana* corespunde unui *camp*. Fiecare camp are un *nume* și un *tip*. Dacă un camp conține o valoare unica atunci el se numeste *cheie primara* (engl. *primary key*). Se pot crea legaturi intre

relatii prin stocarea cheilor primare ale unei relatii intr-o alta relatie sub forma de *chei externe* (engl.*foreign key*).

SQL (engl.*Structured Query Language*) este un limbaj pentru definirea, interogarea și actualizarea bazelor de date relationale. Cateva comenzi SQL foarte folosite sunt select, insert, delete, update, commit, rollback, grant și revoke.

Comenzile commit și rollback sunt utile pentru lucrul cu tranzactii. O *tranzactie* este o secventa de operatii de acces la o baza de date, secventa ce are proprietatea de *atomicitate*. Acest lucru inseamna ca fie secventa nu se executa, fie trebuie executata in totalitate pentru pastrarea consistentei datelor.

O baza de date care lucreaza in regim tranzactional se poate afla in doua stari:

- starea de *autovalidare* (engl.*autocommit*), caz in care orice cerere de actualizare declanseaza automat modificarea conținutului bazei de date.
- starea de *validare manuala* (engl.*manual commit*), caz in care modificarea conținutului bazei de date are loc atunci cand se executa comanda commit. Pana la executarea acestei comenzi, starea bazei de date dinaintea tranzactiei poate fi refacuta cu comanda rollback.

Comenzile grant și revoke permit administratorilor de sistem sa creeze utilizatori și sa le dea respectiv ia drepturi pe urmatoarele patru nivele:

- Nivel global: drepturile globale se aplica la toate bazele de date de pe un anumit server
- Nivelul baza de date: drepturile alocate la acest nivel se aplica tuturor relatiilor unei baze de date.
- Nivelul relatie: drepturile alocate la acest nivel se aplica tuturor coloanelor unei relatii.
- Nivelul coloana: drepturile la acest nivel se aplica individual, coloanelor unei relatii.

Funcțiile unui server de baze de date sunt :

- Interpretarea cererilor SQL primite de la clienti, executia lor și returnarea rezultatelor.
- Optimizarea interogariilor. O interogare SQL se poate executa in mai multe moduri, diferentele intre timpii de raspuns corespunzatori fiecarui mod putand fi mari. Un server va analiza cererea SQL, va examina modul de stocare a relatiilor și va produce intr-un timp rezonabil un plan de executie cat mai eficient.
- Prevenirea erorilor datorate unui acces concurrent la date.
- Detectarea și inlaturarea situatiilor de interblocaj.
- Administrarea accesului controlat la date din motive de securitate.
- Administrarea operatiilor de arhivare/restaurare a bazei de date. Restaurarea bazei de date in cazul unor caderi ale sistemului se realizeaza prin gestiunea unui *jurnal* (engl.*log*) al tuturor tranzactiilor executate.

JDBC este o interfata de programare și o infrastructura pentru accesul la o baza de date relationala. JDBC isi propune sa standardizeze mecanismul de conectare, de trimitere a cererilor, de lucru cu tranzactii, cat și de reprezentare a rezultatelor unei cereri. Programatorul este inasa liber a foloseasca ce dialect SQL doreste, in funcție de serverul de baze de date cu care lucreaza.

JDBC propune o solutie multinivel:

- La nivelul cel mai de sus, o aplicatie Java executa cereri SQL prin intermediul unei interfete de programare definita in pachetul java.sql. Acest pachet nu conține altceva decat definitiile unor interfete Java. Implementarile acestor interfete sunt furnizate separat pentru diversele sisteme de baze de date.
- Aplicatia acceseaza baza de date prin intermediul unui program special numit *JDBC driver manger*.
- Acest program gestioneaza programele specifice de control (engl.*driver*) pentru fiecare sistem de baze de date in parte. Un program de control implementeaza interfata java.sql.Driver.

Principalele interfete definite in pachetul java.sql sunt:

- Driver. Acesta este interfata asociata cu programul de control. Fiecare program de control

va furniza o clasa care implementeaza aceasta interfata. O astfel de clasa pentru MySQL este `com.mysql.jdbc.Driver`, iar pentru o punte JDBC-ODBC este `sun.jdbc.odbc.JdbcOdbcDriver`.

- `DriverManager`. Aceasta clasa gestioneaza programele de control disponibile pentru conectarea la diverse baze de date.
- `Statement`. Este o clasa utila pentru crearea și executarea de cereri SQL.
- `PreparedStatement`. Este o subclasa a clasei `Statement` utila pentru crearea de enunturi SQL parametrizate și care sa poata fi executate eficient.
- `CallableStatement`. Este o subclasa a clasei `Statement` utila pentru executarea de proceduri stocate.
- `ResultSet`. Se utilizeaza pentru reprezentarea multimii de rezultate obținute in urma executiei unei interogari SQL.
- `ResultSetMetaData` și `DatabaseMetaData`. Sunt clase pentru reprezentarea metadatelor referitoare la o multime de rezultate respectiv la o baza de date. Exemple de metadatae sunt: numele atributelor unei relatii, tipurile atributelor unei relatii, diverse caracteristici ale sistemului de baze de date (standardul SQL, dacă suporta sau nu proceduri stocate, etc.)

Pasii de prelucrare la folosirea JDBC sunt urmatoarii:

i) Incarcarea programului de control specific serverului de baze de date. Clasa incarcata conține membrii statici ce creaza o instanta a programului de control și il inregistreaza la gestionarul programelor de control.

```
Class.forName(com.mysql.jdbc.Driver);
```

ii) Definirea URL-ului conexiunii.

```
String dbUrl = "jdbc:mysql://127.0.0.1/magazin";
```

```
String utilizator = "";
```

```
String parola = "";
```

iii) Stabilirea conexiunii.

```
Connection c = DriverManager.getConnection(dbUrl, utilizator, parola);
```

iv) Crearea unui obiect pentru reprezentarea unei cereri SQL.

```
Statement s = c.createStatement();
```

v) Executarea cererii SQL.

```
ResultSet r = s.executeQuery("select isbn,prim_autor,titlu from Carte;");
```

vi) Prelucrarea rezultatelor.

```
while(r.next()) {  
    System.out.println(  
        r.getString("isbn") + ", "  
        + r.getString("prim_autor")  
        + ": " + r.getString("titlu"));  
}
```

vii) Inchiderea conexiunii.

```
c.close();
```

Pentru tratarea erorilor in operatii SQL, JDBC furnizeaza clasa `SQLException`.

Se recomanda ca parametrii unei conexiuni sa fie indicati cu ajutorul unui fisier de proprietati:

```
driver=sun.jdbc.odbc.JdbcOdbcDriver
```

```
url=jdbc:odbc:carte
```

```
utilizator=admin
```

```
parola=admin
```

Dacă o anumita interogare SQL se executa de un numar repetat de ori, atunci este mai eficient sa se foloseasca un *enunt pregatit* (engl.*prepared statement*). In esenta un enunt pregatit este o interogare compilata și optimizata pentru fi executata mai eficient.

Un enunt pregatit este in acelasi timp și un *sablon de interogare*. Un astfel de sablon conține niste parametri specificati prin semnul intrebarii. Valoarea acestor parametri se completeaza inaintea executiei interogarii.

```
String sablon= "update Angajat set salariu=? where id=?";
PreparedStatement cerere =
    conexiune.prepareStatement(sablon);
float[] salarii_noi = getSalariiNoi();
int[] id_angajati = getIdAngajati();
for (int i=0;i< id_angajati.length;i++) {
    cerere.setFloat(1,salarii_noi[i]);
    cerere.setInt(2,id_angajati[i]);
    cerere.execute();
}
```

Metadatele sunt date pentru descrierea altor date. JDBC conține un numar de clase utile pentru extragerea de metadate referitoare la baze de date, multimi de raspunsuri, tabele, programe de control, etc.

Metadatele sunt utile in urmatoarele situatii:

- Pentru a testa dacă sistemul de baza de date are anumite facilitati, a caror lipsa ar putea produce erori. O astfel de facilitate este spre exemplu disponibilitatea procedurilor stocate.
- Dacă nu se cunoaste structura bazei de date.

Tipuri de metadate:

- Metadate referitoare la baza de date. Se obțin cu ajutorul clasei DatabaseMetaData. Aceste metadate includ: numele programului de control, versiunea programului de control, verificarea disponibilitatii standardului SQL2, verificarea disponibilitatii unor facilitati specifice ale SQL, etc.
- Metadate referitoare la multimea de rezultate. Se obțin cu ajutorul clasei ResultSet. Aceste metadate includ: numarul de attribute din multimea de rezultate, numele atributelor, numl relatiei din care s-au obținut attributele, tipurile acestor attribute, etc.

Metadatele pot fi folosite pentru implementarea unor programe utilitare. Un exemplu ar putea fi un program utilitar pentru executarea unei interogari generale și salvarea rezultatelor impreuna cu diverse informații referitoare la baza de date.

Intr-o aplicatie de comert electronic se inglobeaza obligatoriu o baza de date. De asemenea, o astfel de aplicatie va conține și un nivel de obiecte din omeniuil problemei (engl.*business objects*). Se pune problema maparii acestor obiecte in relatii ale modelului relational. Pentru aceasta se creaza cate o clasa corespunzatoare ficarei relatii a bazei de date, cate o variabila membru pentru ficare atribut al acestei relatii și metode corespunzatoare de acces la aceste variabile membru.

Codul metodelor de acces va folosi cereri SQL pentru acces la attributele corespunzatoare din baza de date.

Deschiderea unei conexiuni la o baza de date este un proces consumator de timp. El poate dura mult mai mult decat executarea unei cereri. Din acest motiv se pune problema re folosirii conexiunilor in aplicatiile care se conecteaza de un numar repetat de ori la o baza de date. Se foloseste o clasa speciala numita *rezerva de conexiuni* (engl.*connection pool*). Aceasta clasa ruleaza pe un fir separat și are urmatoarele responsabilitati:

- Prealocarea conexiunilor pe constructorul clasei. Constructorul mai primeste un numar maxim de conexiuni, un numar initial de conexiuni și un indicator dacă rezerva sa astepte sau nu cand se cere o conexiune și nu exista nici una libera.
- Gestiunea conexiunilor disponibile. Dacă se cere o conexiune și exista o conexiune libera, atunci ea este alocata și trecuta in lista conexiunilor ocupate.

- Alocarea de noi conexiuni. Dacă se cere o conexiune, nu exista nici una libera și nu s-a atins numărul maxim de conexiuni, atunci se creaza o noua conexiune pe un fir separat. Dacă insa s-a atins limita maxima, se asteapta sau nu in funcție de indicatorul setat in constructor.
- Asteptarea pentru eliberarea unei conexiuni.
- Inchiderea conexiunilor.

Intr-o aplicatie cu miniservere, rezerva de conexiuni poate fi alocata unui singur miniserver sau se poate partaja intre mai multe miniservere.

5.5. LIMBAJUL DE METAMARCARE XML

XML este un acronim pentru *eXtensible Markup Language* – limbaj extensibil de marcare. XML este un limbaj de meta-marcare - un limbaj pentru definirea limbajelor de marcare. *Limbaj de marcare* (engl.*markup language*) = un limbaj pentru descrierea formei unui document (cum trebuie interpretat conținutul documentului). Un limbaj de marcare definește un *tip de document*. Un document scris într-un limbaj de marcare este compus din *date* și *marcaje*. *Marcajele* sunt acele informații care indica utilizatorului cum trebuie interpretat conținutul documentului (datele propriu-zise).

Exista diverse tipuri de marcaje:

- *Stilistice*: indica stilul de prezentare a documentului: I, U, B, FONT, etc.
- *Structurale*: indica cum este structurat documentul: HEAD, BODY, P, H, etc
- *Semantice*: indica semnificatia conținutului documentului: TITLE, META, etc.

Un limbaj de marcare creat cu XML se numeste și *aplicatie XML* (ex. XHTML).

Utilizarea XML conferă următoarele **avantaje**:

- Faciliteaza interoperabilitatea datelor intre aplicatii.
- Oferă o alternativa la raspandirea *formatelor proprietare*.
- Atat datele cat și marcajele sunt stocate sub forma de text, fapt ce permite manevrarea și editarea usoara, practic cu orice editor de texte.
- Fiind un limbaj de meta-marcare, este configurabil.
- Poate fi citit și interpretat relativ usor de un agent uman.
- Este standardizat, ceea ce inseamna ca este relativ unanim cunoscut și acceptat.
- Este liber in sensul ca specificatia este disponibila liber, este independent de platforma și este bine sustinut de unelte software disponibile liber.

Tipuri de aplicatii care folosesc XML sunt:

- Interactiunea cu doua sau mai multe surse de date, reprezentate in formate diferite = *integrarea datelor* (engl.*data integration*) disponibile de la aplicatii diferite putand rula pe platforme diferite
- Transferarea unei parti a prelucrarilor asupra datelor pe calculatoarele client in arhitecturile client-server
- Furnizarea de vederi diferite asupra acelorasi date
- Agenti pentru colectarea de informații

Documente XML bine-formate

Cerinta de a fi *bine-format* (engl.*well-formed*) este o constrangere de baza impusa documentelor XML. Un document *XML bine-format* trebuie sa respecte regulile de sintaxa incluse in specificatia XML 1.0, adica:

- Documentul conține unul sau mai multe elemente. Un element - un nod al unui arbore.
- Exista un singur element - *radacina* - care conține in interiorul sau celelalte elemente;
- Fiecare element, exceptand radacina, trebuie incuibat corect in cadrul unui element cuprinzator numit *parintele* sau.

Un document XML conține *date* și *marcaje*. Datele sunt reprezentate sub forma unor secvente de caractere (engl.*character data*). Marcajele indica structura documentului (sunt *metadata*). Ele includ:

- Marcajele de inceput și sfarsit ale elementelor;
- Marcajele elementelor vide;
- Comentariile;
- Referintele la entitati;
- Delimitatorii sectiunilor CDATA;
- Definitile de tip de document (engl.*Document Type Definition* – DTD);
- Instructiunile de prelucrare.

Un document XML se compune din trei parti:

- Declaratie; indica faptul ca avem un document XML.
- *Prolog (antet)* - poate fi vid. Conține o definitie de tip sau o referinta la definitia de tip.
- *Elementul radacina*, care conține in interiorul sau toate celelalte elemente.

Un exemplu îl constituie **arborii binari în XML**:

```
<?xml version="1.0" standalone="yes"
  encoding="UTF-8" ?>
<!-- Un arbore binar reprezentat în XML -->
<ARBBIN>
  <INFO>a</INFO>
  <ARBBIN>
    <INFO>b</INFO>
    <ARBBIN/>
  <ARBBIN>
    <INFO>d</INFO>
    <ARBBIN/>
  <ARBBIN/>
</ARBBIN>
<ARBBIN>
  <INFO>c</INFO>
  <ARBBIN>
    <INFO>e</INFO>
    <ARBBIN/>
  <ARBBIN/>
</ARBBIN>
</ARBBIN>
```

Prima linie din programul prezentat este *declaratia* de document XML. Textul inclus între <!-- și --> este un *comentariu*. Elementul de baza pentru structurarea unui document este *elementul*. Un element este inclus între un marcaj de început și un marcaj de sfârșit.

<ARBBIN> este un *marcaj de început* și </ARBBIN> este un *marcaj de sfârșit*.

Primul marcaj <ARBBIN> începe *elementul radacina* al documentului.

<ARBBIN/> este un *marcaj vid* și este echivalent cu <ARBBIN></ARBBIN>. Indica un *element vid*. Elementul <ARBBIN>a</ARBBIN> conține textul "a" în interiorul sau.

Elemente XML cu atribute:

- Informația din cadrul unui nod al arborelui se poate reprezenta ca valoare a unui *atribut*.

...

```
<ARBBIN>
  <INFO info="b" />
  <ARBBIN/>
  <ARBBIN>
    <INFO info="d" />
    <ARBBIN/>
  <ARBBIN/>
</ARBBIN>
</ARBBIN>
```

...

- Un element poate avea mai mult decat un atribut.
<INFO cheie="10" info="b" />

Documente XML valide

Validitatea este o constrangere aditionala care poate fi impusa documentelor XML. Ea se bazeaza pe ideea de *tip de document* mostenita din SGML.

Un document XML se numeste *valid* dacă este bine format și dacă exista o definiție de tip de document – DTD cu care documentul este consistent. Se spune ca documentul *este o instanta* a acestei definiții de tip.

Adaugand o DTD documentului XML care conține un arbore binar se obține:

```
<?xml version="1.0" standalone="yes" ?>
<!-- Un arbore binar reprezentat în XML -->
<!DOCTYPE ARBBIN [
  <!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
  <!ELEMENT INFO (#PCDATA) >
]>
<ARBBIN> ...
```

În acest caz, DTD-ul este conținut în cadrul aceluiași fișier cu documentul XML. DTD-ul este inclus între marcajele <!DOCTYPE ARBBIN []>. Marcajul de start al DTD-ului indica elementul radacina al documentului.

DTD-ul conține o declarație pentru fiecare element al documentului.

Fiecare declarație de element specifica numele elementului și un *model al conținutului* sau. DTD-ul poate fi plasat într-un fișier separat. Dacă presupunem ca DTD-ul este plasat în fișierul *arbbin.dtd*, atunci va trebui sa schimbam doar specificatia DTD-ului din documentul XML:

```
<!DOCTYPE ARBBIN SYSTEM "arbbin.dtd">
```

Fișierul *arbbin.dtd* va conține doar declarațiile elementelor:

```
<!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
<!ELEMENT INFO (#PCDATA)>
```

Dacă informația dintr-un nod se reprezinta ca valoare a atributului *info* al elementului INFO atunci DTD-ul este:

```
<!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
```

```
<!ELEMENT INFO EMPTY>
<!ATTLIST INFO
  info CDATA #REQUIRED>
```

Definiii de tip de document

O definitie de tip de document specifica:

- Elementul radacina al documentului
- Numele, atributele și modelul de conținut al fiecarui element din document:

```
<!ELEMENT nume_element model_de_conținut >
```

Modelul de conținut al unui element se specifica cu !ELEMENT și poate fi:

- *Vid, specificat prin EMPTY*
- *Text, specificat prin #PCDATA*
- *Alte elemente structurate în secvența (,), optionale (?), alternative (/) sau repetitive (* și +)*

Atributele se specifica pentru fiecare element în parte cu !ATTLIST incluzand numele atributului, tipul valorii sale și setarea valorii implicite:

```
<!ATTLIST nume_element
  nume_atribut tip_atribut setare_valoare_implicita
  ...
  nume_atribut tip_atribut setare_valoare_implicita >
```

Tipuri de atribute pot fi: enumerare, CDATA, etc. Setari de valoare implicite pot fi: #IMPLIED, #REQUIRED, #FIXED, etc.

Entități, șiruri de caractere neanalizate, spații de nume

Orice procesor de XML trebuie să distingă între date și marcaje. Pentru a putea desemna caracterele < > ' și " ca date trebuie să folosim *entitățile predefinite* < > ' și ". Pentru & se folosește entitatea predefinită &.

Caracterele speciale UNICODE pot fi referite *&#codZecimal;* sau *odHexazecimal;*. Astfel © se specifica prin © sau ©.

Pentru a indica faptul că un șir de caractere trebuie să nu fie analizat de procesorul documentului XML se folosește o secțiune CDATA:

```
<INFO><![CDATA[Text într-un element <INFO>]]></INFO>
```

Entitățile parametrice sunt asemănătoare cu macroinstrucțiunile și sunt utile pentru organizarea unui DTD.

```
<!ENTITY % nume "lista_nume">
```

Numele de elemente și atribute dintr-un document XML pot defini un *vocabular partajat* destinat reutilizării în cadrul unor aplicații multiple. Pentru evitarea coliziunilor de nume între astfel de vocabulare partajate se folosesc spațiile de nume (engl.*namespaces*).

Limbaje formale și analizoare sintactice

XML este un metalimbaj formal pentru definirea de limbaje formale specifice. Limbaj formal = o mulțime de secvențe de simboluri (enunțuri, fraze) construite pe baza unei mulțimi de reguli de formare numită *sintaxa* limbajului și interpretate potrivit unui anumit înțeles numit *semantica* limbajului.

Pentru analiza și prelucrarea enunțurilor unui limbaj formal (a documentelor XML în

particular) este necesar un program special numit *procesor de limbaj*. O componenta de baza a unui procesor de limbaj este *analizorul sintactic* (engl.*parser*). Un analizor sintactic pentru un limbaj formal *L* primeste la intrare o multime de enunturi și îndeplinește următoarele funcții:

- Decide dacă enunturile verifica sintaxa limbajului *L*.
- Dacă nu, indica cat mai precis posibil ce reguli sunt încălcate și unde anume.
- Dacă da, produce o reprezentare structurata intermediara a enunturilor astfel încat ele sa poata fi ulterior interpretate și prelucrate conform semanticii lui *L*.

Un programator de aplicatii va fi interesat în primul rand cum poate integra un procesor de limbaj în cadrul unei aplicatii. În cazul XML el se va confrunta cu problema integrarii analizoarelor sintactice de XML. În general exista doua modalitati prin care un analizor sintactic poate comunica rezultatele analizei celorlalte componente ale aplicatiei: i) prin evenimente – *analizor sintactic bazat pe evenimente* (engl.*event-driven parser*); ii) prin construirea explicita a unei structuri intermediare care sa descrie complet enunturile analizate – *translator*.

Analiza și prelucrarea documentelor XML folosind Java se face astfel:

- Un analizor sintactic de XML poate fi i) *cu validare* (engl.*validating parser*), caz în care el verifica consistenta documentului XML cu un DTD sau ii) *fara validare* (engl.*non-validating parser*), caz în care el verifica doar dacă documentul XML este bine-format.
- Exista o standardizare a interfetelor de programare (engl.*Application Programming Interface* – API) pe care trebuie sa le implementeze un analizor sintactic de XML astfel încat el sa poata fi integrat cat mai usor și portabil în cadrul unei aplicatii. În prezent exista doua API standard:
 - Simple API for XML – SAX, un standard *de-facto* existent în doua versiuni SAX1 și SAX2. Defineste o multime de interfete pentru un analizor sintactic bazat pe evenimente (engl.*event-driven parser*). SAX a fost definita întâi pentru Java, dar s-au definit ulterior versiuni similare și pentru alte limbaje (C++);
 - Document Object Model – DOM, un standard W3C pentru un analizor sintactic care traduce un document XML într-o structura de date abstracta de tip arbore numita DOM. DOM se doreste a fi o specificatie independenta de limbaj, fiind valabila și pentru limbajele de scripting (JavaScript)
- Se folosește pachetul XML for Java de la IBM (*XMLAJ*). El se bazeaza pe proiectul *xml.apache.org* din cadrul Apache Software Foundation, în speta pachetul *Xerces*.

Standardul SAX

SAX este un standard *de facto* pentru o interfata de programare a unui mecanism de analiza sintactica bazat pe evenimente a documentelor XML. Un programator de aplicatii va putea folosi orice biblioteca de programe de analiza sintactica conforma cu standardul SAX, fiind suficient sa cunoasca doar aceasta interfata standard.

Termenul *bazat pe evenimente* se refera la folosirea unor funcții *call-back*. Aceste funcții trebuie furnizate de programatorul de aplicatii și ele trateaza o serie de evenimente de interes primite de la analizorul sintactic. Un analizor sintactic conform cu SAX va trebui sa implementeze interfata SAX. Funcțiile *call-back* sunt implementate de obiecte speciale de tratare a evenimentelor (engl.*handler*). Un astfel de obiect trebuie sa implementeze niste interfete standard. O biblioteca conforma cu SAX va trebui sa conțină partea de interfete standard SAX și o implementare particulara a acestor interfete. O astfel de bibliotca este *Xerces*.

O interfață SAX conține trei parti:

- Partea de baza *org.xml.sax*. Aici se definesc interfetele standard ale obiectelor de tratare a evenimentelor SAX.
 - *ContentHandler* – *evenimente referitoare la conținut*
 - *DTDHandler* – *evenimente referitoare la DTD*

- *ErrorHandler* – evenimente de eroare
- *EntityResolver* – evenimente de tratare a entitatilor
- *XMLReader* – interfata unui analizor sintactic XML orientat pe evenimente
- *Attributes* – interfata pentru o lista de attribute XML
- Partea de clase de ajutor org.xml.sax.helpers. Aici se definesc niste implementari implicite pentru interfetele SAX.
- *DefaultHandler* – implementare implicita a tuturor celor 4 interfete pentru tratarea evenimentelor SAX.
- *AttributesImpl* – implementare implicita pt o lista de attribute XML
- ...
- Partea de extensii org.xml.sax.ext.

Folosirea SAX presupune parcurgerea urmatorilor pasi:

- Furnizarea de implementari pentru obiectele de tratare a evenimentelor SAX. O metoda comoda este extinderea implementarii implicite din clasa DefaultHandler. Aceasta clasa furnizeaza implementari implicite pentru toate cele 4 interfete standard SAX. În cadrul acestei implmentari se vor redefini doar metodele care sunt necesare.
- Se creeaza o clasa pentru reprezentarea unui analizor sintactic. Pentru aceasta este util sa se utilizeze o biblioteca care sa dispuna deja de o astfel de implementare. Spre exemplu, pachetul *Xerces* dispune de clasa SAXParser.
- În programul principal se realizeaza urmatoarele prelucrari:
 - Se creaza un obiect de tratare a evenimentelor SAX..
 - Se creaza un obiect pentru reprezentarea unui analizor sintactic.
 - Se înregistreaza obiectul de tratare a evenimentelor la analizorul sintactic.
 - Se executa funcția de analiza sintactica a obiectului analizor.
- În funcțiile de tratare a evenimentelor de analiza sintactica se realizeaza toate prelucrarile necesare din cadrul aplicatiei respective. Spre exemplu, se pot folosi aceste evenimente pentru extragerea informațiilor relevante pentru aplicatie din cadrul documentului XML și reprezentarea lor sub forma unei structuri intermediare. Aceasta structura se poate prelucra ulterior.

Interfața DOM

DOM (nivelul 2) este o platforma și o interfata independenta de limbaj ce permite programelor și scripturilor sa acceseze și sa actualizeze dinamic conținutul și structura documentelor XML.

Pachetul de baza org.w3c.dom poate fi gandit ca o specificatie a unui tip abstract de date pentru reprezentarea unui document XML sub forma unui arbore. Principalele interfete definite în acest pachet sunt urmatoarele:

- Node – este tipul primar al unui nod dintr-un arbore DOM
- NodeList – reprezinta o colectie ordonata de noduri
- Element – reprezinta un element al unui document XML Extinde tipul Node.
- NamedNodeMap – reprezinta o colectie de noduri ce pot fi accesate prin nume (de exemplu nodurile atribut unui element XML).
- Document – reprezinta nodul document al unui arbore DOM. Extinde tipul Node.
- Attr – reprezinta un nod atribut al unui element XML. Extinde tipul Node.
- Text –
- Comment –
- Entity –
- ProcessingInstruction –
- ...

La fel ca și pentru SAX, folosirea DOM presupune existența unei biblioteci care să implementeze interfețele DOM. O astfel de bibliotecă este pachetul *Xerces*.

Folosirea DOM presupune parcurgerea următorilor pași:

- Implementarea unui modul de prelucrare a informației dintr-un document XML reprezentat sub forma unui arbore DOM.
- În programul principal se realizează următoarele prelucrări:
 - *Construirea unui obiect pentru reprezentarea unui analizor sintactic conform cu DOM. Pachetul Xerces furnizează pentru aceasta clasa DOMParser.*
 - *Se execută funcția de analiză sintactică a obiectului analizor.*
 - *Se extrage obiectul document și se prelucrează.*

```
DOMParser parser = new DOMParser();  
parser.parse( ... );  
Document doc = parser.getDocument();  
Element root = doc.getDocumentElement();  
...
```

Limbajele XSL și XSLT

XSL (Extensible Stylesheet Language) este un limbaj de prelucrare a documentelor XML. XSL este o aplicație XML.

XSL are două componente relativ distincte:

- *Limbajul de transformare XSLT*. XSLT permite transformarea unui document XML într-un alt document XML sau document text. XSLT 1.0 este o recomandare W3C din 16 Noiembrie 1999. XSLT folosește limbajul XPath pentru adresarea diverselor părți din arborile unui document XML.
- *Limbajul de formatare XML-FO*. XML-FO permite descrierea formatului de prezentare a unui document XML prin furnizarea unui vocabular de formatare.

Pentru reprezentarea unui document XML sunt necesari în general doi pași:

- *Transformarea* documentului XML inițial într-un document XML care folosește vocabularul XML-FO. Acest pas poate fi opțional și o filtrare a conținutului documentului inițial.
- *Formatarea* documentului XML-FO rezultat, cu ajutorul unui program special de formatare, adică transformarea sa într-un format specific de prezentare/ afișare/ vizualizare. Exemple de astfel de formate sunt PDF, Postscript și RTF.

În WWW se poate folosi XSLT pentru a transforma un document XML într-un document XHTML sau WML. Transformarea poate avea loc fie la server, fie la client.

Transformări XSL

XSLT este un *limbaj de transformare* a unui document XML într-un alt format. Formatul de ieșire poate fi tot XML (de exemplu XHTML sau WML) sau alt format text.

Pentru producerea unei transformări este nevoie de trei elemente:

- *Sursa transformării*, documentul XML de intrare
- *Specificatia transformării*, un document XSLT
- *Motorul transformării*, un program numit *procesor XSLT*. Acest program aplică specificatia XSLT a transformării, sursei transformării.

Concepte de bază XSLT:

- Documentul XML furnizat ca intrare a unei transformări XSLT este de forma unui arbore.
- XSLT modelează un document sub forma de arbore, similar cu XPath. XPath este un limbaj de expresii pentru adresarea unei părți dintr-un document XML.
- XPath și XSLT recunosc 7 tipuri de noduri ale arborelui unui document XML de intrare: i)

Radacina documentului: este nodul de start al oricarui document; ii) *Element*: reprezinta un element al sursei XML. iii) *Text*: reprezinta un conținut text dintr-un element al sursei XML. iv) *Atribut*: reprezinta un atribut al unui element din sursa XML; v) *Spatiu de nume*: vi) *Comentariu*: vii) *Instructiune de prelucrare*: reprezinta textul unei instructiuni de prelucrare din sursa XML.

- Pe multimea nodurilor unui document este definita o relatie de ordonare liniara numita *ordinea documentului*. In esenta aceasta ordine exprima urmatoarele: i) nodul radacina este primul; ii) nodurile element apar înaintea copiilor lor; iii) attributele și spatiile de nume asociate unui element apar înaintea copiilor elementului respectiv; iv) spatiile de nume asociate unui element apar înaintea atributelor elementului respectiv.v) ordinea relativa a spatiilor de nume și a atributelor unui element este dependenta de implementare.
- Pentru fiecare nod se definește o *valoare* a nodului. De exemplu, pentru nodul radacina valoarea se obține concatenand conținutul tuturor nodurilor descendente de tip text.

Modelul de prelucrare folosit de XSLT este următorul:

- O transformare XSLT descrie o multime de reguli pentru transformarea unui arbore de intrare într-un arbore de iesire. O *regula* conține o *parte de conditii*, numita *sablon* (engl.*pattern*) și un *tipar* (engl.*template*). Sablonul este identificat cu nodurile arborelui sursa. În cazul unei *identificari* (engl.*matching*), tiparul este instantiat și astfel el genereaza o parte din arborele documentului de iesire.
- Prelucrarea unui arbore sursa se bazeaza pe parcurgerea recursiva a arborilor:
 - Prelucrarea de baza este aplicarea regulilor unei multimi de noduri din arborele sursa. Procesul începe prin aplicarea regulilor listei de noduri formata doar din nodul radacina.
 - Se obține un arbore de iesire prin aplicarea regulilor unui singur nod din arborele sursa. În cazul aplicarii regulilor unei liste de noduri, portiunea care rezulta din arborele de iesire se obține prin concatenarea arborilor obținuti prin aplicarea regulilor fiecarui nod din lista.
 - Aplicarea regulilor unui nod presupune determinarea nodurilor din subarborele sau carora li se poate aplica cel puțin o regula, potrivit ordinii documentului. Dacă pentru nodul selectat se pot aplica mai multe reguli, pentru a determina regula aplicabila se foloseste o strategie de rezolvare a conflictelor bazata pe specificitate. Regula cu partea de conditii este cea mai specifica va fi selectata. Nodul caruia i se aplica regulile devine *nodul curent*, iar lista de noduri din care a facut parte cand fost selectat este *lista nodurilor curente*.
 - Un tipar poate conține instructiuni pentru selectarea altor noduri carora li se vor aplica regulile de transformare. Dacă lista acestor noduri este nevida, atunci procesul de aplicare a regulilor conținea recursiv pentru aceasta lista pana cand nu se mai selecteaza noduri noi pentru prelucrare.

Șabloane în XSL

Un document XSLT conține *reguli de transformare*. O regula de transformare are sintaxa urmatoare:

```
<xsl:template match="sablon">  
  tipar  
</xsl:template>
```

Specificarea sablonului se face cu atributul *match* și se aplică următoarele reguli:

- Identificarea nodului radacina al documentului se face cu sablonul */*.
- Identificarea unui element se face cu un sablon identic cu numele elementului.
- Identificarea unui element arbitrar se face cu sablonul ***.
- Identificarea elementelor copil ale unui anumit element se face utilizand un sablon de forma *parinte/copil*. Mai general, se poate folosi un ablon de forma $e_1/e_2/.../e_n$. El identifica elementele e_n care au un lant de stramosi de forma e_1, e_2, \dots, e_{n-1} . Oricare dintre

- e_i poate fi *. Unş ablon de forma $/e_1/e_2/.../e_n$ identifica un lant de stramosi ce incepe din radacina.
- Identificarea elementelor descendent ale unui anumit element se face cu un sablon de forma *stramos//descendent*.
- Identificarea unui atribut se face cu unş ablon de forma *@atribut*. Identificarea unui nod text, comentariu sau instructiune de prelucrare se face cu sabloane de forma *text()*, *comment()* respectiv *processing_instruction()*.
- Identificarea după unul din mai multe sabloane posibile se face cu un sablon de forma $s_1 | s_2 | \dots | s_n$.

Şabloane cu teste:

- Pe un nod se pot face teste suplimentare adăugând la un şablon un test de forma [*conditie*].
 - Identificarea unui element cu un atribut se face cu un şablon de forma: *element[@atribut]*.
 - Identificarea unui element cu un atribut cu o anumită valoare se face cu un şablon de forma *element[@atribut="valoare"]*.
 - Identificarea unui element *e* prin poziţia sa în lista nodurilor frate care sunt elemente *e* se face cu un şablon de forma *element[pozitie]* sau *element[position()=pozitie]*.
 - Testele se pot încuiba. De exemplu *element[copil[@atribut]]* identifică un element care are un element copil cu un anumit atribut.
 - Într-un test se poate folosi şi operatorul not. De exemplu, şablonul *e[not last()]* identifica toate elementele *e* care nu sunt ultimele în lista nodurilor frate care sunt elemente *e*.
 - Testele pe un nod se pot combina cu caile de noduri pentru a construiş abloane complexe. De exemplu, *e[@a="valoare"]//e₂* identifică elementele *e₂* care sunt descendente ale unor elemente *e₁* cu atributul *a* cu valoarea *v*.

Un exemplu de transformare trivială ce poate fi aplicată oricărui document XML, cu acelaşi rezultat, generarea textului Ceva banal, este următorul:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  version="1.0">
  <xsl:template match="/">
    Ceva banal.
  </xsl:template>
</xsl:stylesheet>
```

xsl:value-of determina valoarea unui nod. Nodul se specifica cu atributul *select*, relativ la nodul curent.

xsl:apply-templates determina aplicarea recursiva a regulilor de transformare unor noduri selectate cu atributul *select*, relativ la nodul curent identificat.

Cel mai simplu exemplu este regula implicita pentru nodul radacina, respectiv pentru fiecare element din documentul sursa:

```
<xsl:template match="/" | "*">
  <xsl:apply-templates/>
</xsl:template>
```

Această regulă are ca efect aplicarea regulilor copiilor nodului radacina şi copiilor tuturor nodurilor element.

xsl:value-of se va folosi direct doar în contexte în care este evident nodul a carui valoare va fi selectata. În cazul în care pot fi selectate mai multe noduri, numai valoarea primului dintre ele va fi generata.

În cazul în care trebuie generate valorile nodurilor selectate, exista doua solutii:

- Generarea unui apel recursiv folosind *xsl:apply-templates*.

```
<xsl:template match="parinte">
```

```

<xsl:apply-templates select="copil"/>
</xsl:template>
<xsl:template match="copil">
  <xsl:value-of select="."/>
</xsl:template>

```

- Folosirea `xsl:for-each` realizand astfel prelucrarea explicita a fiecarui nod selectat.

```

<xsl:template match="parinte">
  <xsl:for-each select="copil">
    <xsl:value-of select="."/>
  </xsl:for-each>
</xsl:template>

```

Expresii XPath

`xsl:apply-templates`, `xsl:value-of` și `xsl:for-each` folosesc atributul `select` pentru selectarea unei multimi noduri ce fac obiectul unor prelucrari. Atributul `select` apare de asemenea și în `xsl:copy-of`, `xsl:variable`, `xsl:param` și `xsl:sort`.

Valoarea atributului `select` este o expresie XPath. Șabloanele de identificare a nodurilor sunt o submultime a multimii expresiilor XPath.

O expresie XPath este de unul din urmatoarele 4 tipuri: multime de noduri, logic, numar, sir de caractere.

O expresie care selectează o multime de noduri se numeste *expresie cale* (engl.*location path*). O cale este compusa dintr-un numar de *pasi* (engl.*location step*). Un pas are un *nod context*. Fiecare subcale selecteaza o multime de noduri relativ la un nod context. Fiecare nod din aceasta multime este nod context pentru pasul urmator.

Un pas se compune din 3 componente:

- O *axa* ce specifica relatia dintre nodurile selectate de pas și nodul context.
- Un *identificator de nod* care specifica tipul și numele nodurilor selectate
- Zero sau mai multe predicat care pot filtra suplimentar multimea de noduri selectate.

BIBLIOGRAFIE

Baze de date

1. Codd, E.F., *The Relational Model for Database Management*, Addison-Wesley, 1990
2. Date, C.J., *An Introduction to Database Systems*, Addison-Wesley, Boston, 2004
3. Dolliner, R., *Baze de Date și Gestiunea Tranzacțiilor*, Editura Albastră, Cluj, 1997
4. Forta, B., *SQL pentru Începători*, (SAMS) Teora, 2002 (traducere Daniel Cihodaru, după ediția din 1999)
5. Fotache, M., *Proiectarea bazelor de date. Normalizare și postnormalizare. Implementări SQL și Oracle*, Ed. Polirom, Iași, 2005
6. Ionescu, F., *Baze de Date Relaționale și Aplicații*, Editura Tehnică, București, 2004
7. Opper, A., *SQL fără mistere*, Ed. Rosetti Educational, București, 2006
8. Peterson, J., *Baze de date pentru începători*, Ed. All, 2003
9. Welling, L., Thomson, L., *Dezvoltarea aplicațiilor cu PHP și MySQL*, Teora, București, 2005
10. *** *Sistemul de gestiune SQL Server*, <http://www.microsoft.com/sql>
11. *** *Sistemul de gestiune MySQL*, <http://www.mysql.com>

Programare orientată obiect și structuri de date

12. C. Bologa, *Algoritmi și structuri de date*, Editura RISOPRINT, Cluj-Napoca, 2005
13. A. Carabineanu, *Structuri de date*, Editura Matrixrom, 2006
14. V. Crețu, *Structuri de date și algoritmi*, Editura Orizonturi universitare, Timișoara, 2000
15. J. Knuth, *Arta programării calculatoarelor*, Editura Teora, 1999
16. I. Ignat, C.L. Ignat, *Structuri de date și algoritmi*, ed. Alabastra, Cluj, 2007
17. I. Ivan, M. Popa, P. Pocatilu, (coordonatori), *Structuri de date. Tipologii de structuri de date*, Editura ASE, 2009
18. G. Soava, *Programare orientată obiect, Teorieși aplicații*, Ed. Universitaria, Craiova, 2010
19. F. Wempfen, *Visual Studio 9*, Editura Teora, București, 2007

Sisteme de operare

20. Acostăchioaie D., *Administrarea și configurarea sistemelor Linux*, ediția a II-a, Ed. Polirom, Iași, 2003
21. Acostăchioaie D., Buraga S., *Utilizare Linux. Noțiuni de bază și practică*, Ed. Polirom, Iași, 2004
22. Cristea V., ș.a., *UNIX*, Ed. Teora, București, 1993
23. Ignat I., ș.a., *Sistemul de operare UNIX. Gestionarea fișierelor.*, Ed. MicroInformatica, Cluj-Napoca, 1992
24. Ignat I., Kacso A., *UNIX – Gestiunea proceselor*, Ed. Alabastră, Cluj-Napoca, 1995
25. Pilat F., *UNIX*, Ed. Teora, București, 1992
26. Tanenbaum A. S., *Modern Operating Systems*, Prentice Hall, 2001
27. Wielsch M., *Le grand livre de UNIX*, Editions MicroApplication, Paris, 1994
28. *****, <http://www.redhat.com>
29. *****, <http://metalab.unc.edu/pub/Linux/docs>

Proiectarea sistemelor informatice

30. Zaharie, D., Rosca, I., *Proiectarea obiectuala a sistemelor informatice*, Dual Tech, Bucuresti, 2002
31. Oprea, D., *Analiza si proiectarea sistemelor informatiionale economice*, Editura Polirom, Bucuresti, 1999

32. Lungu, I., Sabau, Gh. s.a., *Sisteme informatice - Analiza, proiectare si implementare*, Editura Economica, Bucuresti, 2003
33. Stanciu V., *Proiectarea sistemelor informatice de gestiune*, Ed. Cison, București 2000
34. C. J. Date, *An Introduction to Database Systems*, 8th Edition, published by PearsonEducation, Inc. Adison Wesley Higher Education, 2004

Programare Internet

35. T. Anghel, *Programarea in PHP. Ghid practic*, Editura Polirom, 2005
36. F. M. Boian, R. F. Boian, *Tehnologii fundamentale Java pentru aplicatii Web*, Editura Albastra, 2009
37. S.C.Buraga, *Proiectarea siturilor Web. Design si functionalitate*, Editura Polirom, 2006
38. S.Buraga, *Programarea in Web 2.0*, Editura Polirom, 2007
39. S. Buraga, *Tehnologii XML*, Editura Polirom, 2006
40. S. C. Buraga , *Tehnologii Web*, Editura MATRIX ROM, București, 2001
41. Coulouris, G., Dollimore, J, Kindberg, T, *Distributed Systems. Concepts and Design* (3rd ed.), Pearson Education, 2001
42. L.Sângeorzan, C. L. Aldea, *Tehnologii internet*, Ed. Univ. Transilvania, 2003
43. Ceri , Stefano et al, *Designing Data-Intensive Web Applications*, Morgan Kaufmann, 2002
44. S Tudor, V. Huțanu, *Crearea si programarea paginilor WEB*, ed. L&S SOFT, 2006